Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Ein modularer Beweiser für agentenabhängige Terminologien

Diplomarbeit

Leipzig, März 2001

vorgelegt von:

Thomas Hofmann

geb. am: 29.1.1975

Studiengang: Informatik

Kurzzusammenfassung

Ziel der vorliegenden Arbeit ist die Entwicklung eines modular aufgebauten Erfüllbarkeitstesters für verschiedene unimodale und eine polymodale Beschreibungslogik, die zur Repräsentation von agentenabhängig modelliertem terminologischen Wissen genutzt werden kann.
Es werden dazu Tableaukalküle eingeführt, und deren Implementationen vorgestellt. Die Tableaukalküle bearbeiten Beschreibungslogik- und Modallogikanteile getrennt, dieser Trennung folgt dann auch der Aufbau des Testers. Zur Bearbeitung der Beschreibungslogikanteile wird das System FaCT benutzt, welches als Modul in den Erfüllbarkeitstester integriert wird. Eine notwendige Vorverarbeitungstufe wird von einem weiteren Modul realisiert.
Für die Performance des gesamten modularen Systems spielt die interne Kommunikation eine entscheidene Rolle. Deshalb ist eine optimierte Variante der Kommunikation zwischen FaCT und dem Gesamtsystem entwickelt worden.
Während des Tests verschiedener Formeln eines Benchmarks konnte das hier entwickelte modulare System im Vergleich mit anderen Systemen akzeptable Leistungen erbringen.

Danksagung

Diese Arbeit wurde im Rahmen des DFG-Projekts Wo582/3-1 *„Kombination von Modal- und Beschreibungslogik und ihre Anwendung zur Repräsentation intensionalen und dynamischen Wissens “* angefertigt.

Bei den folgenden Personen möchte ich mich ganz besonders für die Unterstützung der vorliegenden Arbeit bedanken:

- Herrn Dr. F.Wolter für die Vergabe der Aufgabenstellung und die intensive Betreuung der Arbeit,
- Herrn Prof. G.Brewka für die freundliche Unterstützung der Arbeit,
- Herrn H.Sturm und meinem Kommilitonen Dietmar Schulz für die gute Zusammenarbeit und zahlreichen Hinweise zur Arbeit und
- meinem Kommilitonen Alexander Bluhm für seine technische Unterstützung.

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Wissensrepräsentation

Wer kennt nicht das Sprichwort:

„*Wissen ist Macht !* “

Aber was ist Wissen überhaupt ? Wo findet man Wissen ? Und wie kann man Wissen verwalten ? All diese Fragen sind bis heute nicht vollständig aufgeklärt. Auch diese Arbeit soll keine Patentlösung dieser Fragen liefern, es soll lediglich eine Möglichkeit der Repräsentation von Wissen vorgestellt werden.
Wissen beginnt mit dem Sammeln und Verknüpfen von Fakten. Das Internet zum Beispiel bietet heute auf der Schwelle zur Informationsgesellschaft eine riesige Ansammlung von Informationen. Aber was nützt schon die grösste Datenmenge zur Lösung einer sehr spezifischen Frage. Ohne Ordnung, einer Struktur innerhalb der Informationen, gleicht jede Frage der sprichwörtlichen Suche einer Nadel im Heuhaufen. Deshalb sind Datenmengen fast immer durch verschieden Strukturen geordnet. Viele Informationen findet man zum Beispiel in Datenbanken gespeichert. Letztere sollen hier nicht Gegenstand der Diskussion sein, da sie die Aufgabe haben, Daten lediglich zu verwalten.
Vielmehr interessiert hier die Frage nach neuen Strukturen zur Wissensrepräsentation und deren Nutzung. Zum Beispiel beschäftigen sich viele Teilgebiete der künstlichen Intelligenz ausführlich mit Problemen der Darstellung von Informationen einerseits und andererseits mit der Gewinnung von Informationen aus der gewählten Darstellung. Anders gesprochen werden Fakten über ein bestimmtes Wissensgebiet gesammelt und Regeln entworfen wie man neue Fakten ableiten kann.
Formal kann eine solche Wissensrepräsentation mittels einer Logik

beschrieben werden. Die Eigenschaften der verwendeten Logik gestalten dann die Möglichkeiten zur Wissensrepräsentation.
Je ausdrucksstärker die Logik ist, desto komplexere Wissenszusammenhänge können dargestellt werden.

1.2 Beschreibungslogik

Viele Computerapplikationen, die in ihren Programmen auf Daten aller Art zurückgreifen, benutzen und verändern diese Informationen innerhalb von Modellen, die den Anwendungsumgebungen zugrundeliegen. Eine Möglichkeit der Darstellung einer solchen Umgebung in einfacher symbolischer Form ist die Beschreibung der Objekte innerhalb des Modells sowie die Darlegung vorhandener Beziehungen zwischen den Objekten.
Hilfreich bei der Erklärung der untereinander bestehenden Objektbeziehungen ist die Zuordnung von Objekten mit gleichen bzw. zumindest ähnlichen Eigenschaften zu Klassen und der Aufbau einer Klassenhierarchie, die auf den entsprechenden Sub- bzw. Superklassenbeziehungen innerhalb der Modellwelt beruht. Siehe hierzu [29].
Beschreibungslogik stellt nun solch einen Formalismus zur Wissensrepräsentation dar. Beispiele dieser Verwendung findet man in [14] oder auch in [11].

Der Gebrauch von Beschreibungslogik in Wissensrepräsentationssystemen hat zur Entwicklung vieler *DLKRS* (engl. Description Logic Knowledge Representation System) geführt. Das erste dieser Systeme war KL-ONE [18], welches dann zu Weiterentwicklungen bzw. Implementierungen von neuen Systemen den Anstoß gab. Zu diesen Systemen gehören NIKL [23], KRYPTON [16], KRIS [17], SB-ONE [19], CycL [20], LOOM [21], K-rep [22], CLASSIC [26], BACK [25], CANDIDE [24], GRAIL [27], FaCT [1] and DLP [30]. Eine historische Zusammenfassung all dieser Systeme findet man in [28].

Die Wissensmodellierung innerhalb der Beschreibungslogik geschieht mittels Konzepten [1] , durch die Klassen modelliert werden. Konzepte können zusätzlich noch durch Rollen [2] miteinander in Beziehung gebracht werden. Und einzelne Objektnamen [3] können verwendet werden, um auszudrücken, dass ein Objekt Instanz eines

[1] ... entsprechen einstelligen Prädikaten
[2] ... entsprechen zweistelligen Prädikaten
[3] ... entsprechen somit den Elementen des Modellbereichs (Domain)

Konzeptes ist.
Modellierungen beginnen dabei mit atomaren Konzepten und Rollen, und definieren durch Verknüpfungen mittels vorgeschriebener Operatoren komplexere Konzepte.
Am Beispiel von ALC (siehe [10]), einer in der Beschreibungslogik weit verbreiteten Sprache, werden im folgendem die Möglichkeiten der Modellierung vorgestellt. Neben ALC gibt es aber noch eine Vielzahl anderer Modellierungssprachen der Beschreibungslogik (siehe [12]).
Zur Bildung von komplexeren Konzepten in ALC sind

1. die bekannten aussagenlogischen Junktoren als Operatoren zulässig. D.h. sind C und D Konzepte, dann auch $C \wedge D$, $C \vee D$ und $\neg C$. Und

2. sind zwei Rollenkonstrukte zulässig. D.h. sei R sei eine Rolle und C ein Konzept, dann sind $\exists R.C$ und $\forall R.C$ ebenfalls Konzepte.

Die Bedeutung der aussagenlogischen Junktoren ist klar (als Konjunkton, Disjunktion sowie Negation), und der Ausdruck $\exists R.C$ beschreibt die Klasse/das Konzept, zu dessen Objekten jeweils mindestens ein Objekt existieren muss, das der Klasse/dem Konzept C angehört und mit diesem in der Beziehung/Rolle R steht.
Der Ausdruck $\forall R.C$ wird als Abkürzung für den Ausdruck $\neg\exists R.\neg C$ gebraucht.

Sind zum Beispiel innerhalb einer Programmumgebung alle Personen von Interesse, benutzt man das nicht weiter spezifizierte Konzept `Person`. Danach kann man die Gruppe der Eltern definieren indem man fordert: Eltern sind Personen, die Kinder haben. Dazu benutzt man das Konzept `Elternteil` und die Rolle `hat-Kind`:

$$\texttt{Elternteil} = \texttt{Person} \wedge \exists \texttt{hat-Kind}.\texttt{Person} \ .$$

Konkrete Personen können ebenfalls berücksichtigt werden, indem man zum Beispiel festlegt:

1. *Tom* und *Jerry* sind beides Instanzen des Konzepts `Person` und

2. als Paar eine Instanz der Rolle `hat-Kind`.

Damit erfüllt *Tom* auch die Bedingungen um ein `Elternteil` (Instanz dieses Konzeptes) zu sein.

Diese Beispiele demonstrieren zwei verschiedene Arten, Wissen darzustellen, erstens terminologische Beziehungen zwischen den Konzepten und Rollen und zweitens direkte Zuordnungen von Objekten zu Konzepten und Rollen. Deshalb trennen fast alle $DLKRS$ ihre Fakten der Wissensbasis in eine terminologische Komponente (TBox genannt) und eine assertorische (ABox genannt).
Die TBox wird von einem $DLKRS$ benutzt, um mittels eines Algorithmus Subsumtionsbeziehungen zwischen den Konzepten zu berechnen. Die erhaltenen Subsumtionen werden dann zum Aufbau einer Konzepthierarchie verwendet. Desweiteren kann eine TBox innerhalb eines $DLKRS$ noch zum Test der Erfüllbarkeit eines Konzepts bezüglich dieser TBox sowie eines Konsistenzchecks verwendet werden.
Die ABox kann zum Berechnen von *retrieval-* und *realisation-*Aufgaben benutzt werden. Die erste Verwendung behandelt die Frage nach vorhandenen Instanzen eines gegebenen Konzepts. Die zweite Aufgabe die Suche nach dem spezifischsten Konzept, zu dem eine gegebene Instanz noch gehört.
Voraussetzung für diese Anwendungen ist die Existenz eines vollständigen, korrekten und praktikablen Algorithmus für den Subsumtionstest [1]. Dieser Test kann aber auf die Frage nach der Erfüllbarkeit eines Konzepts zurückgeführt werden. Deshalb beschäftigt sich diese Arbeit insbesondere mit dem Aufbau eines Erfüllbarkeitstesters.

1.3 Modallogik

In der jüngsten Zeit spielt das Internet in vielen Computeranwendungen eine immer stärkere Rolle. Besonders für Programme der Wissensrepräsentation stellt es eine riesige Informationsquelle dar. Um diese Flut an unüberschaubaren Fakten zu bewältigen, das heißt zu verwalten und insbesondere nach interessantem Inhalt zu sortieren, sind Multi-Agenten-Systeme von Interesse. Man stellt sich dazu zum Beispiel mehrere selbstständig das Internet durchsuchende Agenten vor, die Fakten sammeln und bei einem persönlichen dem Nutzer angepasstem Assistenten ablegen. Dieser regelt dann die endgültige Auswahl betreffender Fakten, und steuert die weitere Suchaufgaben der einzelnen Agenten.
Dieses Bild eines Multi-Agenten-Systems zum Sammeln von Wissen ist jedoch nur eine Vorstellung einer möglichen Anwendung der Modellierungssprache wie sie hier entworfen wird. Welchen Anteil

Modallogik an dieser Sprache hat, soll nun im folgendem dargelegt werden.

Besonders in Systemen, in denen mehrere intelligente Agenten gestellte Aufgaben kooperativ lösen sollen, ist es unabdingbar „Wissen über Wissen " zu verwalten.
D.h. ein jeder Agent eines Multiagentensystems sollte über die ihm bekannten Fakten informiert sein und desweiteren Annahmen über Fakten, die anderen Agenten bekannt sind, treffen können. Dazu ist es nützlich, Fakten als Wissen eines bestimmten Agenten kennzeichnen zu können, um damit Annahmen über das Wissen verschiedener Agenten treffen zu können.

Einen geeigneten Formalismus zur Modellierung solcher Fragestellungen stellt Modallogik dar, wie sie zuerst in Hintikka's[8] Arbeit eingeführt wurde. Hierbei besteht die Möglichkeit, auf aussagenlogische Formeln Modaloperatoren anzuwenden. Das heißt während der Modellierung kann auf die Kenntnis über das Wahrsein von Sätzen Bezug genommen werden. Dies geschieht indem man Formeln/Fakten als *bekannt*, *gewusst*, *geglaubt* usw. kennzeichnet. Einführende Beispiele hierzu finden sich im 3. Kapitel in [32].
Diese Erweiterung der klassischen Logik durch epistemische Modalitäten führt zur Verwendung von Wissens- bzw. Glaubensoperatoren in der Modellierungssprache. So kann man zum Beispiel mit der Formel $\Box_{Agent_i}\varphi$ zum Ausdruck bringen, dass Agent Nummer i den Fakt φ glaubt oder, bei entsprechend anderer Interpretation der modalen Operatoren, Agent i besitzt das Wissen φ (usw.).

Unter Modallogik wird meistens eine Menge Γ von Formeln (der modalen-aussagenlogischen Sprache) verstanden, die

1. alle aussagenlogischen Tautologien,
2. das K-Axiom: $\Box(p \to q) \to (\Box p \to \Box q)$ und
3. weitere unabhängige Axiome (um verschiedene Modallogiken zu erhalten) enthält, sowie
4. abgeschlossen ist unter folgenden drei Regeln:

MP Wenn φ und $\varphi \to \psi$ in Γ sind, so auch ψ in Γ.

SUB Ist φ in Γ und erhält man φ' durch Substitution aus φ, dann ist φ' in Γ.

NEC Ist φ in Γ, so ist auch $\Box_i\varphi$ in Γ.

Dieser Aufbau der Modallogik erweitert den Aufbau der klassischen Aussagenlogik. Neu dabei ist das Axiom K und die Regel *NEC*.

Schränkt man sich im Kontext von epistemischen Modaloperatoren auf Wissensoperatoren ein, steht die Regel *NEC* für die Eigenschaft:

> aus der Gültigkeit einer Formel folgt das Wissen dieser Formel.

Das Axiom K steht dann für die Folgerung:

> aus dem Wissen, dass p und dass aus p q folgt,
> folgt das Wissen q.

Diese Annahmen führen zur logischen Allwissenheit. Diese stellt eine starke Idealisierung der Realität dar, denn wer ist schon wirklich in der Lage *alle(!)* Konsequenzen und Schlussfolgerungen seines Wissens zu kennen. Aber für Computeranwendungen, gerade im Hinblick auf Multi-Agenten-Systeme, stellt diese Idealisierung, sofern die Agenten nur mit begrenztem Wissen umzugehen haben, kein Hindernis dar.

Die zuvor angedeuteten Möglichkeiten der Interpretation der Modaloperatoren hängen direkt von der Auswahl der zusätzlich zum Axiom K zugelassenen Axiome ab. Die wichtigsten dabei sind:

D	$\Box\varphi \rightarrow \neg\Box\neg\varphi$,
T	$\Box\varphi \rightarrow \varphi$,
4	$\Box\varphi \rightarrow \Box\Box\varphi$ und
5	$\neg\Box\varphi \rightarrow \Box\neg\Box\varphi$.

Wie auch das K-Axiom, haben die Axiome D, T, 4 und 5 eine Lesart, in der der Modaloperator als Wissens- bzw. Glaubensoperator verwendet wird. Damit verbinden sich folgende intuitive Prinzipien, die mit Hinzunahme des Axioms zur Eigenschaft der betreffenden Modallogik werden:

> für das Axiom D: Aus dem Wissen von φ folgt, dass das Gegenteil von φ nicht bekannt ist, denn es nicht wahr, dass $\neg\varphi$ gewusst wird.
>
> für das Axiom T: Aus dem Wissen von φ folgt, dass φ wahr ist.

für das Axiom 4: Diese Axiom behandelt das unter dem Namen positive Introspektion bekannte Prinzip. Es fordert, dass aus dem Wissen von φ, auch das Wissen vom Wissen von φ folgt.

für das Axiom 5: Schliesslich handelt dieses Axiom von der negativen Introspektion. Und damit wird gefordert: Wenn φ nicht bekannt ist, dann wird auch gewusst, dass φ nicht bekannt ist.

Durch diese und andere hinzugenommene Axiome entstehen natürlich eine Vielzahl unterschiedlicher Modallogiken, die einfach durch die Namen der betreffenden Axiome bezeichnet werden. So wird zum Beispiel die Modallogik, die durch oberes Schema und die Axiome D , 4 und 5 entsteht, kurz $KD45$ genannt. Historisch haben sich für verschiedene Logiken noch andere Namen durchgesetzt so werden die Logiken KT nur als T, $KT4$ als $S4$ und $KT45$ als $S5$ bezeichnet.

Die semantische Grundlage der Logiken bilden Kripke Semantiken (auch Mögliche-Welten-Semantik oder Relationssemantik genannt). Dabei werden Formeln in relationalen Strukturen $\langle W, R_1, ..., R_n \rangle$ interpretiert. Die Elemente aus W werden hierbei als mögliche Welten aufgefaßt und wR_iw' heißt, dass der Agent i in der Welt w die Welt w' für möglich hält. Damit ist in der Welt w die Aussage $\Box_i p$ wahr genau dann, wenn p in allen Welten, die i für möglich hält, wahr ist, d.h. p ist wahr in allen Welten aus $\{v \in W \mid wRv\}$. Entsprechend ist die Aussage $\Diamond_i p$ wahr in w genau dann, wenn es eine Welt v gibt, die Agent i in w für möglich hält und in der p wahr ist, d.h. p gilt in einer Welt aus $\{v \in W \mid wRv\}$.

Jedes der oben genannten Axiome steht in diesem Kontext auch für eine Eigenschaft der Relation, die die Welten untereinander verbindet:

D	- Serialität,
T	- Reflexivität,
4	- Transitivität und
5	- Euklidizität .

Eine allgemein akzeptierte Möglichkeit zur Modellierung der Modaloperatoren mit den genannten Axiomen als Wissens- (engl. knowledge operator) bzw. (engl. belief operator) Glaubens-Operatoren gibt es nicht. Siehe auch [9]. Deshalb müssen für

konkrete Anwendungen entsprechende Varianten der Kombination verschiedener Axiome ausgewählt werden, um eine passende Modellierung zu finden. Jede mögliche Axiomenkombination modelliert dabei unterschiedlichste Aspekte der Wissensrepräsentation.

In dieser Arbeit wird $S5$ als Basis-Modallogik benutzt. Deshalb sollte man sich bei der Benutzung der Modaloperatoren zur Modellierung diverser Fakten bewusst sein, welche Axiome gültig sind und ob diese mit der gewünschten Modellierung verträglich sind. Zum Beispiel ist das Axiom T bei der Verwendung von Glaubens-Operatoren oft nicht wünschenswert. Denn wenn ein Agent etwas glaubt, muss es nicht automatisch wahr sein (T-Axiom: $\Box\varphi \rightarrow \varphi$).

Eine geeignete Modellierung für Multi-Agenten-Systeme verlangt für jeden Agenten einen eigenständigen Modaloperator. Diese Operatoren haben jeweils die gleichen Eigenschaften und sind voneinander unabhängig. Technisch wird jeder Modaloperator deshalb getrennt betrachtet.
Man spricht in diesen Fällen von einer polymodalen Version der betreffenden Modallogik. Für den in dieser Arbeit vorgestellten Beweiser/Erfüllbarkeitstester ist deshalb $S5$ in der polymodalen Version $S5_n$ verwendet worden.

1.4 Kombination von Modallogik und Beschreibungslogik

In den letzten beiden Abschnitten sind zwei unterschiedliche Ansätze zur Wissensrepräsentation vorgestellt worden. Einerseits Modallogik, die sehr gut zur Modellierung intensionaler Gegenstandsbereiche geeignet ist, und andererseits Beschreibungslogik, welche sich eher zur Darstellung extensionaler und statischer Zusammenhänge eignet.
So kann mit letzterer zum Beispiel recht einfach definiert werden, was man unter dem Begriff einer untreuen Frau verstehen kann:

(a) $Frau = Person \wedge Weiblich$

(b) $Untreue\_Frau = Frau \wedge \exists hat.Liebhaber$

usw.

Mit Hilfe von Modallogik können nun durch Einsetzen von einfacheren Fakten (hier zum Beispiel (b) und (c)) komplexere Aussagen

modelliert werden:

(c) $\Box_{(Agent\ A\ glaubt)}(b) \wedge \neg\Box_{(Agent\ B\ glaubt)}(b)$

(d) $\Box_{(Agent\ C\ glaubt)}(c)$

usw.

Dieses Beispiel soll verdeutlichen in welcher Weise statische Fakten dynamisch verwaltet werden können, indem Konzeptkonstrukte in Beschreibungslogik als Grundelemente von komplexeren Formeln in Modallogik dienen. Die Dynamik der Darstellung wird hierbei durch die Modellierung der Intension eines Agenten erreicht. So zeigt der Ausdruck (d) zum Beispiel wie verschiedene Auffassungen über einen Begriff dargestellt werden können. Versucht man diese Formel nur mit Beschreibungslogik darzustellen, ist dies ohne Inkonsistenzen nicht möglich.
Das nächste Beispiel soll den Versuch, unsicheres Wissen eines Agenten zu modellieren, darstellen. Dabei wurden um den Ausdruck kompakt darstellen zu können der oben genannte Punkt (b) und die Abkürzung $\Box_X$ für $\Box_{(Agent\ X\ glaubt)}$ verwendet.

(e) $\Box_B(\neg\Box_A(b) \wedge \neg\Box_A\neg(b))$.

Dieser Ausdruck zeigt das Wissen des Agenten B über die Unsicherheit des Agenten A bezüglich des Wissens einer bestimmten Aussage (b).
Trotz kleiner Einschränkungen an die Lesbarkeit der Ausdrücke zeigen diese Beispiele wie gut die Kombination von Beschreibungs- und Modallogik zur Modellierung von Multi-Agenten Systemen geeignet ist. Weitere Beispiele solcher Systeme und deren Verbindung zur Modallogik findet man zum Beispiel im 4.Kapitel (Knowledge in Multi-Agent Systems) in [7].

Diese Arbeit ist Teil eines Projekts[4], dass sich mit der Implementierung verschiedener modaler Beschreibungslogiken befasst. Die Grundlage eines Wissensrepräsentationssystems auf Basis einer modalen Beschreibungslogik ist ein Erfüllbarkeitstest, der mit einem in akzeptabler Zeit berechenbaren Algorithmus entschieden werden kann.
Ziel des Projekts ist es einen modular aufgebauten Tester für die Erfüllbarkeit einer Formel der entsprechenden modalen Beschreibungslogik zu implementieren.
Eine modulare Aufbauweise wird deshalb angestrebt, da zur

[4]DFG-Projekt Wo582/3-1 *„Kombination von Modal- und Beschreibungslogik und ihre Anwendung zur Repräsentation intensionalen und dynamischen Wissens "*

Implementierung des neue Testers auf bestehende Systeme zurückgegriffen werden soll. Konkret sollen einzelne möglichst unabhängige Kompenenten einerseits den Anteil der Beschreibungslogik und andererseits den der Modallogik verarbeiten.

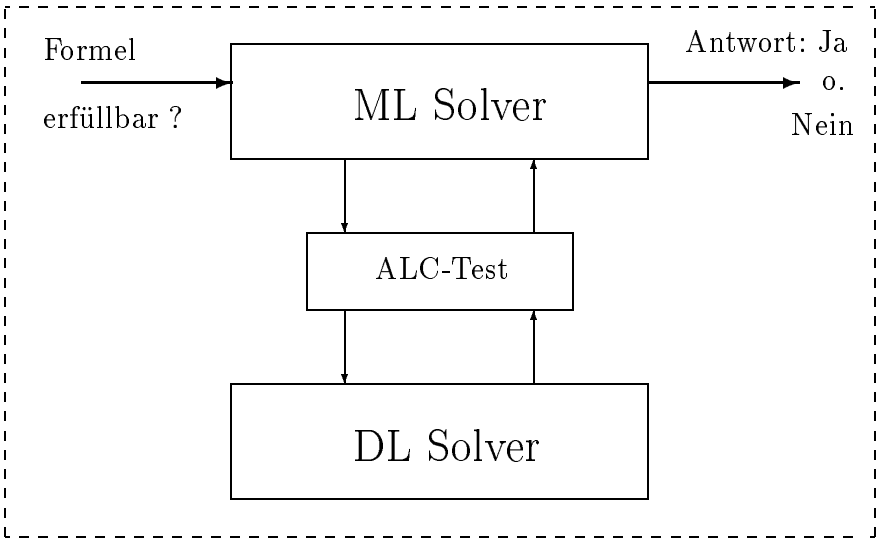

Abbildung 1.1: Aufbau des Beweisers

Wie im Bild dargestellt wird während der Arbeit des Testers, der die modallogischen Komponenten abarbeitet (der ML[5]-Solver), mittels des ALC-Tests, auf ein eigenständiges System, das sich um die beschreibungslogischen Teile kümmert (der DL[6]-Solver), zugegriffen. Betont werden sollte besonders die Eigenständigkeit des DL-Solvers, der unter Umständen eventuell durch verschiedene schon implementierte Beschreibungslogiksysteme realisiert werden kann.

[5] ML für Modallogik bzw. (engl.) modal logics
[6] DL für (engl.) description logics

Kapitel 2

Formale Grundlagen modaler Beschreibungslogiken

2.1 Syntax

Wie bereits während der einführenden Beschreibung der Kombination von Modallogik und Beschreibungslogik angedeutet, sollen hier modale Beschreibungslogiken basierend auf der Beschreibungslogik ALC betrachtet werden. Desweiteren soll es sich bei den zu erweiternden Modallogiken um $K, T, S4$ und insbesondere um den polymodalen Fall von $S5$ (kurz $S5_n$) handeln. Alle modalen Beschreibungslogiken, welche im folgendem besprochen werden sollen, basieren auf der Sprache M^n_{ALC} .

- die gemeinsam verwendeten Symbole von M^n_{ALC} :
  - endlich viele Konzeptnamen : $C_1, ..., C_n$
  - endlich viele Rollennamen : $R_1, ..., R_m$
  - logische Symbole zur Bildung komplexer Konzepte : $\sim, \sqcap, \sqcup, \top, \exists, \forall$
  - sowie entsprechende Klammerzeichen, mit der üblichen Gebrauchsweise (d.h. vereinfachte Klammerung mittels unterschiedlicher Bindungsstärke)
- logische Symbole zur Bildung komplexer Formeln in M^n_{ALC} :

  $=, \neg, \wedge, \vee, \Diamond_i, \Box_i$

Die Menge der ALC-Konzepte ist rekursiv wie folgt definiert :

- Jeder Konzeptname C_i ist ein ALC-Konzept.
- $\top$ ist ein ALC-Konzept.

- Sind C und D ALC-Konzepte und ist R ein Rollenname, so sind $\sim C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$, $\forall R.C$ ALC-Konzepte.
- Nichts sonst ist ein ALC-Konzept.

Analog wird jetzt die Menge der M^n_{ALC} -Formeln definiert :

- Sind C und D ALC-Konzepte, so ist $C = D$ eine (atomare) M^n_{ALC} -Formel , die auch als (atomare) Gleichung bezeichnet wird.
- Sind ψ und ϕ M^n_{ALC} -Formeln , so sind auch $\neg\psi$, $\psi \wedge \phi$ und $\psi \vee \phi$ M^n_{ALC} -Formeln .
- Ist ψ eine M^n_{ALC} -Formel, so sind auch $\Box_i\psi$ und $\Diamond_i\psi$ M^n_{ALC} -Formeln .
- Nichts sonst ist eine M^n_{ALC} -Formel .

Durch Verwendung der atomaren Konzepte *Auto*, *Radio* und *Klimaanlage* und der Rolle *Ausstattung* sind folgende Gleichungen Beispiele für atomare Formeln:

a_1 : gutes-Auto = Auto $\sqcap \exists$ Ausstattung.Klimaanlage

a_2 : gutes-Auto = Auto $\sqcap \exists$ Ausstattung.Radio

a_3 : gutes-Auto = Auto $\sqcap \exists$ Ausstattung.Klimaanlage $\sqcap$ $\exists$ Ausstattung.Radio

Durch agentenabhängige Kombination der Formeln können dann komplexere Zusammenhänge dargestellt werden. Die Formel

$$\Box_{Agent_1} a_1 \wedge \Box_{Agent_2} a_2 \wedge \Box_{Agent_3} a_3$$

sagt zum Beispiel, daß $Agent_1$ ein Auto mit Klimaanlage für ein gutes Auto hält, $Agent_2$ hingegen ein Radio als Teil eines guten Autos für wichtig hält, und $Agent_3$ braucht beides, um von einem guten Auto sprechen zu können.

Falls zum Aufbau von Formeln nur die Modaloperatoren $\Box_i$ und $\Diamond_i$ für ein gewisses i verwendet werden, kann auf den Bezeichner i der Modaloperatoren verzichtet werden.

Im folgenden wird, wenn nichts anderes gesagt wird, davon ausgegangen, dass sich ALC-Konzepte und auch M^n_{ALC} Formeln in Negationsnormalform (kurz NNF) [1] befinden, d.h.:

[1] Eine Umwandlung in NNF ist mittels der üblichen deMorganschen Regeln und der Äquivalenzen $\neg\Box_i\neg\varphi = \Diamond_i\varphi$ und $\neg\Diamond_i\neg\varphi = \Box_i\varphi$ immer möglich.

1. Konzeptnegationen (Symbol : $\sim$) höchstens vor Konzeptnamen und $\top$ auftreten sowie
2. Formelnegationen (Symbol : $\neg$) nur vor atomaren Formeln, also Gleichungen $C = D$.

Desweiteren werden nur noch Gleichungen der Form $C = \top$ berücksichtigt. Denn jede Gleichung der Form $C = D$ lässt sich in folgender Gleichung ausdrücken : $(C \sqcap D) \sqcup (\sim C \sqcap \sim D) = \top$.

2.2 Semantik

Im letzten Abschnitt sind die syntaktischen Grundlagen der modalen Beschreibungslogiken dargelegt wurden. Unterschiede zwischen den einzelnen Logiken treten erst innerhalb der Semantik und später auch bei den Kalkülen auf.
Zur Definition der Semantik der modalen Beschreibungslogiken wird eine auf M^n_{ALC} -Frames basierende Kripke-Semantik eingeführt, die im folgenden definiert werden.

Ein M^n_{ALC} -Frame ist ein Paar $F := \langle W, S \rangle$, bestehend aus einer nichtleeren Menge W und einer Menge S von (zweistelligen) Relationen $\prec_i$ auf W (d.h. $S = \{\prec_1, ..., \prec_r\}$) .

Ein M^n_{ALC} -Modell Ω ist ein Tripel der Gestalt $\langle W, S, I \rangle$, wobei die ersten beiden Komponenten einen M^n_{ALC} -Frame ergeben und I eine Interpretationsfunktion ist , welche jedem $w \in W$ ein ALC-Modell $I(w) := \langle \Delta_w, R_1^{I,w}, ..., R_n^{I,w}, C_1^{I,w}, ..., C_m^{I,w} \rangle$ zuordnet.
Ein ALC-Modell I ist eine Struktur mit folgendem Aufbau : $I = \langle \Delta, R_1^I, ..., R_n^I, C_1^I, ..., C_m^I \rangle$, wobei

- Δ eine nichtleere Menge ist ,
- R_i^I für $i \in \{1, ..., n\}$ zweistellige Relationen auf Δ sind , (d.h. $R_i^I \subseteq \Delta \times \Delta$,)
- C_j^I für $j \in \{1, ..., m\}$ Teilmengen von Δ sind , (d.h. $C_j^I \subseteq \Delta$) .

Im oberen Fall ist dabei ein jedes ALC-Modell, als Teilstruktur innerhalb des Aufbaus eines M^n_{ALC} -Modells , von der jeweiligen Welt w des Gesamtmodells Ω abhängig.

Die Interpretation (Extension) eines komplexen Konzepts C erfolgt dann rekursiv wie folgt :

Sei Ω ein M^n_{ALC} -Modell und $w \in W$, dann

$$\top^{I,w} := \Delta_w \ ,$$

$$(C \sqcap D)^{I,w} := C^{I,w} \cap D^{I,w} \ ,$$

$$(C \sqcup D)^{I,w} := C^{I,w} \cup D^{I,w} \ ,$$

$$(\sim C)^{I,w} := \Delta_w \backslash C^{I,w} \ ,$$

$$(\forall R.C)^{I,w} := \{a \in \Delta_w \mid \forall b \in \Delta_w(\langle a, b\rangle \in R^{I,w} \Rightarrow b \in C^{I,w})\} \ ,$$

$$(\exists R.C)^{I,w} := \{a \in \Delta_w \mid \exists b \in \Delta_w(\langle a, b\rangle \in R^{I,w} \ \& \ b \in C^{I,w})\} \ ,$$

Die letzten sechs Zeilen beschreiben die konkrete semantische Charakterisierung von ALC-Konzepten durch entsprechende ALC-Modelle. Dies deckt sozusagen semantisch den syntaktischen Teil der ALC-Konzepte innerhalb der hier betrachteten Sprache ab. Atomare und komplexere Formeln werden dann durch
M^n_{ALC} -Modelle weiter charakterisiert.

Die semantische Relation der Wahrheit einer Formel σ in einem M^n_{ALC} -Modell Ω an einer Welt w wird nun induktiv über den Aufbau von σ definiert :
Für die Fälle σ atomar bzw. durch die Junktoren $\neg$, $\wedge$ und $\vee$ aufgebaut :

$$\Omega, w \models C = D \text{ gdw } C^{I,w} = D^{I,w}$$

$$\Omega, w \models \neg\phi \text{ gdw } \Omega, w \not\models \phi$$

$$\Omega, w \models \phi \wedge \psi \text{ gdw } \Omega, w \models \phi \text{ und } \Omega, w \models \psi$$

$$\Omega, w \models \phi \vee \psi \text{ gdw } \Omega, w \models \phi \text{ oder } \Omega, w \models \psi$$

und für die modalen Fälle $\sigma = \Box_i\phi$ und $\sigma = \Diamond_j\phi$:

$\Omega, w \models \Box_i\phi$ gdw
für alle $w' \in W$ gilt, wenn $w \prec_i w'$, dann $\Omega, w' \models \phi$

$\Omega, w \models \Diamond_j\phi$ gdw
es ein $w' \in W$ gibt, so dass $w \prec_j w'$, dann $\Omega, w' \models \phi$

Eine Formel φ heisse erfüllbar genau dann, wenn es ein
M^n_{ALC} -Modell Ω und eine Welt w gibt, so dass $\Omega, w \models \varphi$. Eine Formelmenge Φ heisse (simultan) erfüllbar genau dann, wenn es ein M^n_{ALC} -Modell Ω und eine Welt w gibt, so dass $\Omega, w \models \varphi$ für jedes $\varphi \in \Phi$ gilt.

Bisher zeigen weder Syntax noch Semantik Ansätze um von mehreren modalen Beschreibungslogiken sprechen zu können. Um nun von verschiedenen Logiken ausgehen zu können sind noch andere Eigenschaften zur Beschreibung der jeweiligen Logik heranzuziehen:

1. kann man an die Erreichbarkeitsrelationen zwischen den Welten eines Modells Forderungen stellen. Zum Beispiel bei der Zielsprache dieser Arbeit $S5_n$ wird gefordet, dass alle Relationen $\prec_i$ Äquivalenzrelationen sind. Das heißt sie müssen reflexiv, transitiv und symmetrisch sein. Und

2. die Anzahl der Modaloperatoren verändern. Letzteres führt zu den uni- bzw. polymodalen Beschreibungslogiken, das heißt innerhalb der Syntax werden unterschiedliche Anzahlen von Modaloperatoren zugelassen. Beim Aufbau von unimodalen Logiken sind dabei nur $\Box_i$ und $\Diamond_i$ [2] für ein festes i als Modaloperatoren zugelassen. Polymodale Logiken lassen folglich mehr als zwei Modaloperatoren zu.

Was genau ein L-M^n_{ALC}-Frame ist, richtet sich wiederum nach der Spezifikation der Modallogik L. Alle Eigenschaften der Frames die unabhängig von der konkret betrachteten Logik sind, legt dabei die oben eingeführte Definition fest. Zusätzlich wird ein solcher Frame noch durch die Eigenschaften der Erreichbarkeitsrelationen $\prec_i \in S$ genauer charakterisiert.

2.3 Modale Constraintsysteme (MCS)

Alle Tableaukalküle, die im nächsten Abschnitt behandelt werden sollen, basieren auf sogenannten modalen Constraintsystemen (MCS).
Ein solches MCS ist eine (endliche) Menge von modalen Constraints, die einerseits die Aufgabe haben Formeln zu repräsentieren und andererseits auch unterschiedliche Welten darzustellen. Ein modaler Constraint tritt deshalb in zwei möglichen Formen auf:

Form a : ein Ausdruck der Gestalt $\varphi \mid l$, wobei φ eine Formel aus der Sprache M^n_{ALC} ist, oder

Form b : ein Ausdruck der Gestalt $l <_i l'$.

[2] Man beachte den gleichen Bezeichner der Modaloperatoren, deshalb wird dieser auch meist weggelassen.

Die Variablen l und l' werden dabei als sogenannte Weltenvariablen benutzt, welche auch als Label bezeichnet werden. Innerhalb von Constraintsystemen wird also jeder Formel ein Label zugeordnet. Damit wird einerseits jede Formel an eine Welt gebunden (durch Constraints der Form a) und andererseits eine Struktur zwischen den einzelnen Welten/Label (durch Constraints der Form b) festgelegt.

Sei $\mathcal{M}$ ein MCS und $\Omega := \langle W, S, I \rangle$ ein M^n_{ALC}-Modell.

Ω erfüllt $\mathcal{M}$ genau dann, wenn es eine Funktion α von der Menge der in $\mathcal{M}$ vorkommenden Label in die Menge W gibt, so dass α folgenden Bedingungen erfüllt:

1. wenn $\varphi \mid l \in \mathcal{M}$, dann $\Omega, \alpha(l) \models \varphi$,
2. wenn $l <_i l' \in \mathcal{M}$, dann $\alpha(l) \prec_i \alpha(l')$.

Zur besseren Visualisierung von MCS kann man auch sogenannte strukturierte MCS ($sMCS$) heranziehen. Diese gewinnt man aus MCS durch Sortieren der modalen Constraints der Form a anhand ihrer Label. Wobei jedes Label eine zu ihm gehörige Teilmenge erzeugt. Deshalb wird für jede Weltenvariable l aus einem gegebenen MCS $\mathcal{M}$ die Menge

$T_l = \{\phi \mid l \parallel \phi \mid l \in \mathcal{M}\}$ definiert.

Danach werden die so gewonnenen Teilmengen untereinander mittels der modalen Constraints der Form b in Beziehung gestellt. Alle Constraints der Form b bilden somit eine Kantenmenge K . Und formal ist das zu $\mathcal{M}$ gehörige $sMCS$ $\mathcal{M}'$ dann das Paar (T, K) , wobei

$T = \{T_l \mid \text{l kommt in } \mathcal{M} \text{ vor } \}$

und wie oben angedeutet, K die Menge der $l \prec_i l'$ aus $\mathcal{M}$ ist.

Zweck dieser Weltenstrukturen ist es, die konkrete Struktur der modalen Constraintsysteme zu verdeutlichen.

Zur graphischen Darstellung eines $sMCS$ werden alle Mengen T_i als Ellipsen gezeichnet. Im Inneren der Ellipsen (Ovale oder ähnliche Formen) werden alle Formeln der entsprechenden Menge T_i aufgeführt. Die Kanten des $sMCS$ dienen zum Verbinden der Ellipsen untereinander zu einer baumartigen Struktur. Die Knoten/Ellipsen dieses Baumes sind noch durch die entsprechenden

Label gekennzeichnet.

Sei zum Beispiel

$$\mathcal{M} := \{\Diamond p \wedge \Diamond q \mid l_1 \ , \Diamond p \mid l_1 \ , \Diamond q \mid l_1 \ ,$$
$$l_1 < l_2 \ , l_1 < l_3 \ , p \mid l_2 \ , q \mid l_3\}$$

ein MCS . Dann ist $\mathcal{M}'$ das aus $\mathcal{M}$ gewonnene $sMCS$, wobei $\mathcal{M}'$ folgende graphische Darstellung besitzt :

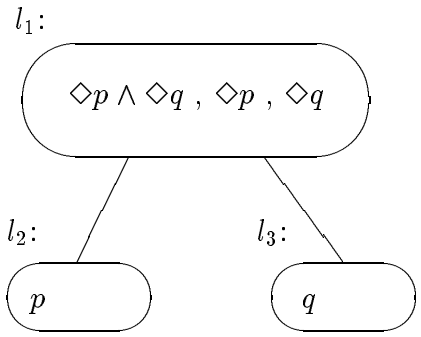

Abbildung 2.1: Beispiel eines $sMCS$

Man beachte bei diesem Beispiel, dass die Kanten keinen Bezeichner tragen. Dies ist möglich da im Beispiel eine unimodale Syntax, das heißt es sind nur $\Box$ und $\Diamond$ als Modaloperatoren zugelassen, benutzt wurde.
Verwendung werden diese $sMCS$ zur graphischen Darstellung von Tableauregeln finden. Diese Regeln beschreiben unter anderem die Arbeitsweise der Tableaus, die im folgendem Kapitel eingeführt werden sollen. Desweiteren ist nicht schwer zu erkennen, das diese Konstrukte den entsprechenden Modellen der Kripke-Semantik entsprechen, sofern der Tableaubeweis vollständig und mit positivem Ergebnis abgelaufen ist.

Kapitel 3

Tableaukalküle

Diese Arbeit basiert auf den in der Beschreibungs- und Modallogik weit verbreiteten Tableaukalkülen. Die Arbeit von R. Goré [2] bespricht sehr viele Logiken und die dazugehörigen Tableaukalküle. Alle Regeln der hier verwendeten Tableaukalküle basieren auf Tableausystemen dieser Arbeit.

3.1 Die Modallogiken $K, T, S4$

3.1.1 Eigenschaften von MCS

Im folgenden sollen verschiedene Tableaukalküle (kurz Tableaus) für die Modallogiken K, T und $S4$ eingeführt werden. Dabei sind mit Modallogiken natürlich die entsprechenden modalen Beschreibungslogiken gemeint, die in diesen Fällen alle eine unimodale Syntax besitzen.
Die im letztem Kapitel eingeführten MCS dienen zusammen mit einigen Regeln als Basis für die Tableaukalküle. In diesen Kalkülen werden zwei Arten von Regeln unterschieden :

1. die nichtmodalen Regeln : $\rightarrow_\wedge$, $\rightarrow_\vee$,
2. die modalen Regeln : $\rightarrow_\Box$, $\rightarrow_\Diamond$

und zusätzlich gibt es noch zwei Tests :

1. der Blocking-Test und
2. der ALC-Test.

Diese Regeln werden auf die modalen Constraints der Form a eines MCS angewandt. Dabei produzieren die Regeln neue modale Constraints, die dem MCS hinzugefügt weden. Constraints der Form b werden nur als Bedingungen für die Anwendung bzw.

Nichtanwendung von Regeln benutzt.

Im folgendem sei S ein MCS.

S ist *vollständig* bzw. heisse *komplett* genau dann, wenn keine der nichtmodalen oder modalen Regeln mehr auf einen Constraint in S angewandt werden kann.

S heisse l *-komplett* genau dann, wenn keine der nichtmodalen Regeln oder modalen Regeln auf einen modalen Constraint der Form a mit dem Label l in S angewandt werden kann. Man spricht dann auch davon, dass das Label l bzw. die entsprechende Welt *vollständig entfaltet* ist.

S heisse *clashfrei* genau dann, wenn S kein Paar modaler Constraints der Gestalt $\neg\varphi \mid l$ und $\varphi \mid l$ enthält.

S heisse *ALC-konsistent* genau dann, wenn für jedes in S vorkommende Label l der ALC-Test auf $S(l)$ [1] ein positives Ergebnis liefert.

Ziel des Tableaus ist es ein komplettes, clashfreies und ALC-konsistentes MCS zu erzeugen.

Anders ausgedrückt bedeutet das, mit Hilfe des Tableaus wird versucht, für eine gegebene Formel φ ein MCS $\mathcal{M}$ zu konstruieren, so dass $\mathcal{M}$ in einem M^n_{ALC}-Modell erfüllbar ist, und damit, falls die Konstruktion gelingt, die Erfüllbarkeit von φ gezeigt wird.

3.1.2 Die nichtmodalen Regeln

Die zwei Regeln $\rightarrow_\wedge$ und $\rightarrow_\vee$ erklären, was mit Konjunktion und Disjunktion von Teilformeln geschieht. Umgangssprachlich formuliert werden bei Anwendung von $\rightarrow_\wedge$ die einzelnen Konjunktionsglieder dem MCS hinzugefügt.
Die Anwendung von $\rightarrow_\vee$ bewirkt das Hinzufügen eines Disjunktionsgliedes zum MCS. D.h. es besteht eine Wahlmöglichkeit zwischen den einzelnen Disjunktionsgliedern.

- Falls $S = S' \cup \{\varphi \wedge \psi \mid l\}$ dann bewirkt die Regel $\rightarrow_\wedge$

  $S' \cup \{\varphi \wedge \psi \mid l\} \rightarrow_\wedge S' \cup \{\varphi \wedge \psi \mid l\ ,\ \varphi \mid l\ ,\ \psi \mid l\}$
  sofern $\varphi \mid l \notin S'$ oder $\psi \mid l \notin S'$ gilt.

[1] ... siehe Abschnitt 3.1.5

- Falls $S = S' \cup \{\varphi \vee \psi \mid l\}$ und weder $\varphi \mid l \in S'$ noch $\psi \mid l \in S'$ gilt, dann ist

  die 1. Möglichkeit der Anwendung dieser Regel:

  $$S' \cup \{\varphi \vee \psi \mid l\} \rightarrow_{\vee} S' \cup \{\varphi \vee \psi \mid l\ ,\ \varphi \mid l\}\ ,$$

  und die 2. Möglichkeit der Anwendung dieser Regel:

  $$S' \cup \{\varphi \vee \psi \mid l\} \rightarrow_{\vee} S' \cup \{\varphi \vee \psi \mid l\ ,\ \psi \mid l\}\ .$$

Die Demonstration der nichtmodalen Regeln mit Hilfe eines strukturierten MCS , wobei Γ eine Formelmenge sei:

- für die Konjunktion

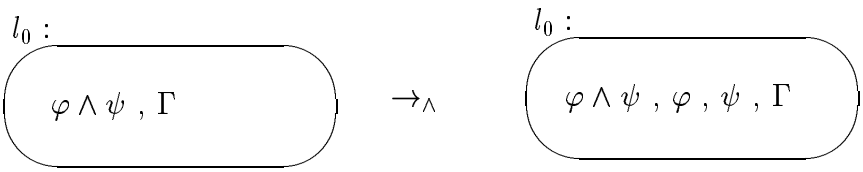

Abbildung 3.1: $\rightarrow_{\wedge}$-Regel

- für die Disjunktion

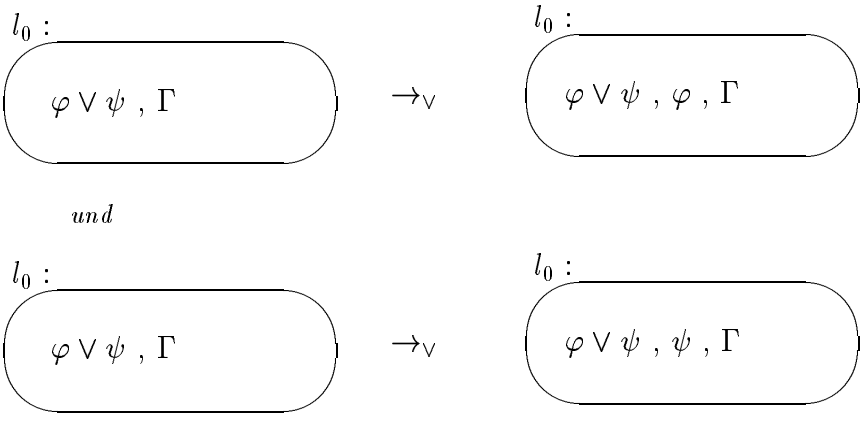

Abbildung 3.2: $\rightarrow_{\vee}$-Regel

3.1.3 Die modalen Regeln

Die zwei Regeln $\rightarrow_\Box$ und $\rightarrow_\Diamond$ treten bei Formeln der Form $\Box\varphi$ bzw. $\Diamond\varphi$ in Aktion. Innerhalb der Syntax sind hier nur die Modaloperatoren $\Box$ und $\Diamond$ zugelassen. Deshalb wird in den Regeln auch keinerlei Bezug auf eventuell vorhandene Bezeichner dieser Operatoren genommen.
Die $\rightarrow_\Diamond$ Regel führt eine neue Welt mit der Formel φ ein.
Die andere Regel erledigt dann das Hinzufügen von φ und auch $\Box\varphi$ in Form von modalen Constraints zu den entsprechend erreichbaren Welten. Dabei werden bei verschiedenen Modallogiken ($K, T, S4$) auch unterschiedlich viele Formeln bzw. Constraints hinzugefügt.

- Falls $S = S' \cup \{\Diamond\varphi \mid l\}$ und

  1. l' ein neues Label ist, d.h sonst nicht in S vorkommt und
  2. für alle Label l'' mit $l < l'' \in S'$ gilt $\varphi \mid l'' \notin S'$,

  dann:

  $$S' \cup \{\Diamond\varphi \mid l\} \rightarrow_\Diamond S' \cup \{\Diamond\varphi \mid l\ ,\ l < l'\ ,\ \varphi \mid l'\}\ .$$

- Für die Modallogik K:
  Falls $S = S' \cup \{\Box\varphi \mid l\}$ und

  1. $\varphi \mid l' \notin S'$ und
  2. $l < l' \in S$

  dann:

  $$S' \cup \{\Box\varphi \mid l\} \rightarrow_\Box S' \cup \{\Box\varphi \mid l\ ,\ \varphi \mid l'\}\ .$$

- Für die Modallogik T:
  Falls $S = S' \cup \{\Box\varphi \mid l\}$ und

  1. $\varphi \mid l' \notin S'$ oder $\varphi \mid l \notin S'$ und
  2. $l < l' \in S$

  dann:

  $$S' \cup \{\Box\varphi \mid l\} \rightarrow_\Box S' \cup \{\Box\varphi \mid l\ ,\ \varphi \mid l'\ ,\ \varphi \mid l\}\ .$$

- Für die Modallogik $S4$:
  Falls $S = S' \cup \{\Box\varphi \mid l\}$ und

  1. $\varphi \mid l' \notin S'$, $\Box\varphi \mid l' \notin S'$ oder $\varphi \mid l \notin S'$ und
  2. $l < l' \in S$

  dann:

  $$S' \cup \{\Box\varphi \mid l\} \rightarrow_{\Box} S' \cup \{\Box\varphi \mid l\ ,\ \varphi \mid l\ ,\ \varphi \mid l'\ ,\ \Box\varphi \mid l'\}\ .$$

Die Abbildungen 3.3, 3.4, 3.5 und 3.6 demonstrieren die modalen Regeln mit Hilfe strukturierter MCS. Dabei seien die Γ_i Formelmengen.

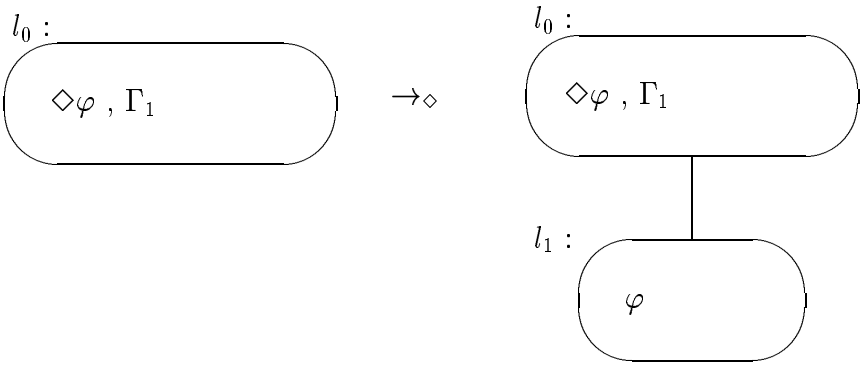

Abbildung 3.3: $\rightarrow_{\Diamond}$-Regel

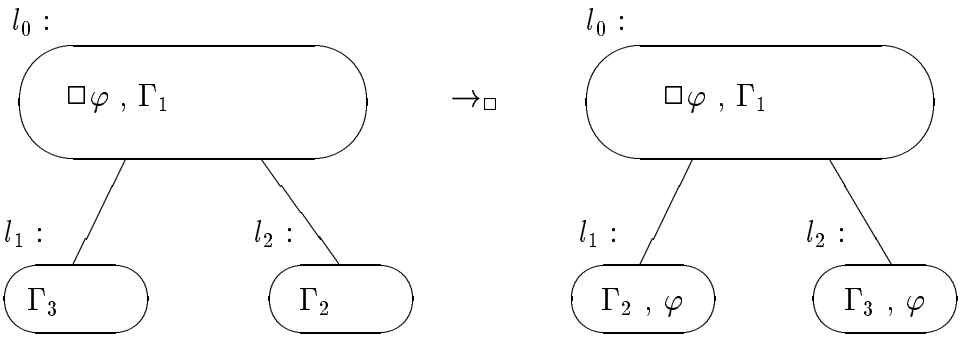

Abbildung 3.4: $\rightarrow_{\Box}$-Regel für die Logik K

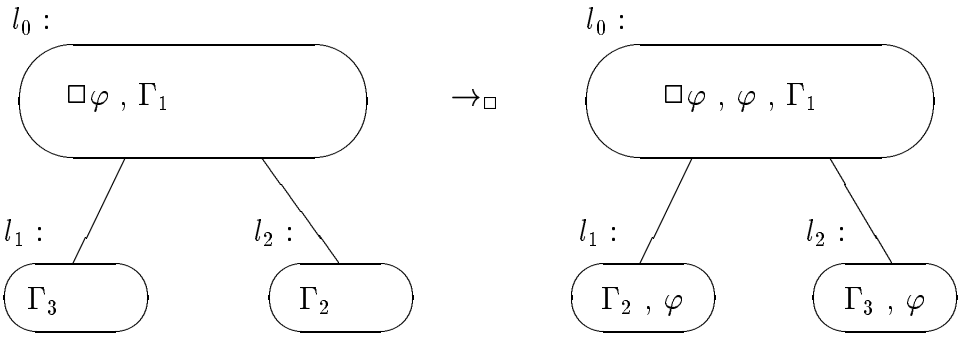

Abbildung 3.5: $\rightarrow_\Box$-Regel für die Logik T

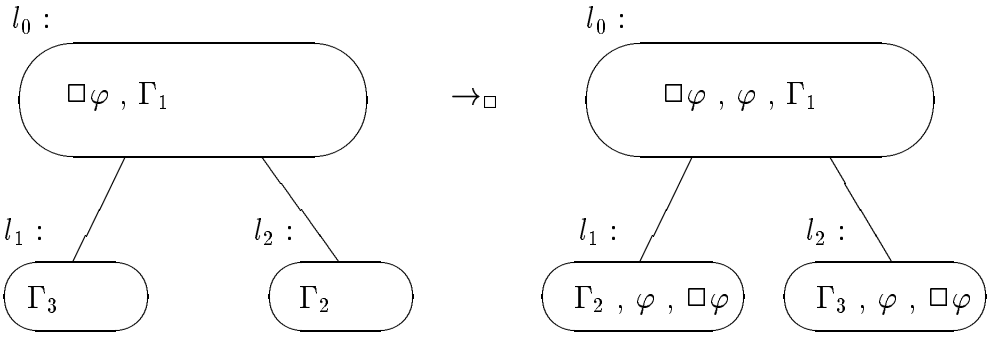

Abbildung 3.6: $\rightarrow_\Box$-Regel für die Logik $S4$

3.1.4 Der Blocking-Test

Dieser Test wird gebraucht um die Terminierung des Tableaus zu garantieren. Er entscheidet darüber, ob eine Welt blockiert werden kann, d.h. diese Welt innerhalb ihrer Vorgängerwelten eine mit den gleichen Eigenschaften wie sie findet. Damit wird einerseits die Terminierung der Konstruktion der Weltenstruktur garantiert und andererseits schon vorhandene Welten wiederverwendet.
Dazu wird ein Prädikat $blocked(\Diamond\varphi \mid l, S, l')$ eingeführt, welches angibt wann ein Label l ein anderes Label l' innerhalb eines MCS S blockiert.

$blocked(\Diamond\varphi \mid l, S, l')$ gdw

1. S ist l-komplett ,
2. $\varphi \mid l \in S$,
3. es existiert eine natürliche Zahl n mit $n \geq 1$, $l' = l_0$, $l = l_n$ und
   $l_0 < l_1 \in S$, $l_1 < l_2 \in S$, ... , $l_{n-1} < l_n \in S$ sowie
4. $\{\Box\psi : \Box\psi \mid l \in S\} \subseteq \{\Box\psi : \Box\psi \mid l' \in S\}$

Während des Ablauf des Algorithmus wird der Blocking-Test auf ein MCS S und ein gerade aktuelles (in Bearbeitung befindliches) Label l_i angewandt.
Getestet wird dabei ob es ein l_j gibt, so dass $blocked(\Diamond\varphi \mid l_j, S, l_i)$ gilt.

3.1.5 Der ALC-Test

Wie bereits in der Einführung erwähnt, ist es Ziel ein modulares System zu schaffen. Dieser Test ist nun innerhalb der modallogischen Ebene zuständig für die ALC-Teile des Systems. Genauer gesagt ist es der Test ob eine Menge von ALC-Konzepten erfüllbar ist. Angewendet wird der ALC-Test auf eine Menge $S(l)$, wobei S ein MCS und l ein Label ist, welches in S vorkommt.

Dabei ist $S(l)$ dann definiert als

$$S(l) := \{C = \top : C = \top \mid l \in S\} \cup \{C \neq \top : C \neq \top \mid l \in S\} ,$$

($C \neq \top$ entspricht dabei $\neg(C = \top)$).

$S(l)$ besteht also aus Gleichungen ($C = \top$) und Ungleichungen ($C \neq \top$). Und der Test versucht dafür ein ALC-Modell zu konstrieren, indem alle Konzepte (alle C) der Gleichungen als TBox in das

ALC-System geladen werden und bezüglich dieser TBox dann nach der Erfüllbarkeit der Konzepte aller Ungleichungen gefragt wird.

3.1.6 Die Terminierung der Regelanwendung

Um ein Ergebnis bei der Anwendung der Regeln zu erhalten, ist es wichtig zu einem kompletten MCS zu gelangen. Dies ist schon deshalb wichtig, um nach einer endlichen Zeitspanne die Entscheidung treffen zu können, ob nun eine Formel erfüllbar ist oder nicht. Das bedeutet es muß einen Zustand geben, wo keine Regel mehr benutzt werden kann, und damit die Anwendung der Regeln terminiert.
Aber es ist leicht zu sehen das z.Bsp. ein MCS S mit
$S := \{\Box\Diamond\varphi \mid l_0\}$ in $S4$ mit der wiederholten Anwendung von $\rightarrow_\Box$ und $\rightarrow_\Diamond$ nie stoppen würde. Denn es würde abwechselnd eine neue Welt durch die Regel $\rightarrow_\Diamond$ eingeführt, und in diese durch die Regel $\rightarrow_\Box$ der Constraint, der für das erneute Einführen neuer Welten verantwortlich ist, übergeben.
Deshalb ist es notwendig insbesondere nach Einführung neuer Welten, zu testen, ob diese Welten in ihrer Funktion durch andere schon vorhandene Welten ersetzt werden können.
Das führt dann im Falle der Ersetzbarkeit einer neuen Welt, was auch als Blockieren dieser Welt bezeichnet wird, zum Abbruch der Bearbeitung der möglichen Nachfolger dieser Welt.
Das ist auch der Grund, warum zum Tableau ein sogenannter *Blocking*-Mechanismus, der durch den Blocking-Test ausgelöst werden kann, gehört.

3.1.7 Das eigentliche Tableauverfahren

Die Erfüllbarkeit einer Formel φ soll durch den Tableau entschieden werden. Deshalb startet der Algorithmus mit einem MCS S, welches nur einen einzigen modalen Constraint enthält, d.h. $S := \{\varphi \mid l_0\}$. Dieser Constraint ist die Formel φ, die zusätzlich an ein Startlabel l_0 gebunden ist.
Ziel des Tableaus ist es, S mittels der Regeln bezüglich aller in S auftretenden Label zu vervollständigen, und dabei zu beachten, dass S clashfrei und ALC-konsistent bleibt.
Deshalb ist die Hauptstrategie des Algorithmus, S nacheinander bezüglich aller vorkommender Label l_i zu vervollständigen. Dazu wendet man Regeln nur auf Constraints mit gleichem Label (zuerst l_0) an, bis S l_0-vollständig ist, prüft ob dieses Label blockiert

werden kann[2] und testet[3] anschließend noch ob die Menge $S(l_0)$ ALC-konsistent ist. Sind die beiden Tests erfolgreich gelungen, also Blocking-Test negativ und ALC-Test positiv, fährt man iterativ mit dem nächsten Label l_1, welches als direkter Nachfolger innerhalb von S ($l_0 < l_1 \in S$) ausgewiesen ist, fort, bis für alle in S vorkommenden Label Vollständigkeit erreicht ist.
Alle Label erreicht man dabei mittels einer *depht-first*-Traversierungsstrategie durch die baumartige Struktur. Der Blocking-Test verhindert dabei unendliche Strukturen, und der ALC-Test stellt für jede Welt ALC-Konsistenz sicher. Deshalb wird durch diese Abarbeitung aller Label stückweise die Weltenstruktur für ein Modell, welches die Anfangsformel erfüllt, aufgebaut.

Kann ein Label blockiert werden folgt daraus, dass alle möglichen Nachfolger dieses Label bei der Bearbeitung keine Berücksichtigung mehr finden. Das heißt alle Constraints die Nachfolgerlabel diese Label enthalten müssen aus dem MCS entfernt werden und anschliessend kann mit der Bearbeitung fortgefahren werden. Ist der Blocking-Test für ein Label negativ ausgefallen, kann mit dem ersten Nachfolgerlabel (falls vorhanden) fortgefahren werden.

Ist der ALC-Test für ein Label positiv wird normal fortgefahren. Bringt der ALC-Test dagegen ein negatives Ergebnis wird im Tableau zurückgesprungen. Das bedeutet das MCS wird in den Zustand direkt vor der letzten Anwendung der $\rightarrow_\vee$ Regel zurückversetzt. Und diese Regel wird dann erneut angewandt aber mit anderem Ergebnis [4] , d.h. Auswahl der anderen Anwendungsmöglichkeit.

Diese Vorgehensweise, d.h. das Backtracking zur letzten $\rightarrow_\vee$ Regel, wird auch benutzt sobald im Tableau ein Clash entdeckt wird. Überprüft, ob ein Clash (d.h. in S gibt es ein Paar $\varphi \mid l_i$ und $\neg\varphi \mid l_i$) vorhanden ist oder nicht, wird dabei bei jeder Regel, die das MCS um neue Constraints erweitert.
Um nun S bezüglich eines Labels l_i korrekt zu vervollständigen, ist es notwendig bezüglich dieses Labels die nichtmodalen Regeln zuerst anzuwenden, danach (wieder nur bezüglich des Labels) alle möglichen $\rightarrow_\Diamond$ Regeln und abschliessend noch die entsprechenden $\rightarrow_\Box$ Regeln (aber auch nur bezüglich des Labels).

[2]... mittels Blocking-Test
[3]... mittels ALC-Test
[4]Denn diese Regel besitzt als einzige mehrere Möglichkeiten der Anwendung.

3.2 Die polymodale Version von $S5$ ($S5_n$)

3.2.1 Die Unterschiede zum letzten Tableau

Der folgende Tableaukalkül für $S5_n$ basiert, wie die schon eingeführten Tableaus, auf Regeln und Tests, die auf ein MCS angewendet werden. Da es jetzt aber Ziel der Modellierung ist, eine polymodale Logik als modale Beschreibungslogik einzuführen, entstehen gerade bei den modalen Regeln Unterschiede. Denn schon innerhalb der Syntax gibt es anstatt der Operatoren $\Box$ und $\Diamond$, eine Anzahl von Operatoren $\Box_i$ und $\Diamond_j$, wobei $1 \leq i \leq n_i$ und $1 \leq j \leq n_j$ gilt. Diese grössere Anzahl von Modaloperatoren führt auch zu einigen Unterschieden innerhalb der Semantik und auch in den MCS. Innerhalb der MCS haben die modalen Constraints der Form b zum Beispiel jetzt die Form $l <_i l'$ (anstatt $l < l'$) .
Wie zuvor sind die Bestandteile des neuen Tableaus:

1. die nichtmodalen Regeln $\rightarrow_\wedge$ und $\rightarrow_\vee$,
2. die modalen Regeln $\rightarrow_{\Box_i}$ und $\rightarrow_{\Diamond_j}$,
3. der ALC-Test sowie
4. der Blocking-Test.

Dabei sind die nichtmodalen Regeln und auch der ALC-Test unveränderte Komponenten aus den letzten Tableaus. Die modalen Regeln und der Blocking-Test sind aber entsprechend angepasst.
Im Ablauf der Anwendung der Regeln gibt es durch die neuen modalen Regeln ebenfalls Neuerungen.

3.2.2 Die modalen Regeln

Beide modalen Regel ($\rightarrow_{\Box_i}$ und $\rightarrow_{\Diamond_j}$) werden wie gewohnt auf modale Constraints, deren Formel als Hauptjunktor/Operator $\Box_i$ bzw. $\Diamond_j$ besitzen, angewandt.
Der Unterschied zu den Regeln des letzten Tableaus besteht in der Abhängigkeit von den jeweiligen Bezeichnern (i bzw. j) der Modaloperatoren. Dies führt bei der $\rightarrow_{\Diamond_j}$ Regel wie gewohnt zur Einführung einer neuen Welt, die aber abhängig von j mit den anderen Welten verbunden ist.
Und die $\rightarrow_{\Box_i}$ Regel erweitert nur vom aktuellen Label erreichbare Welten des MCS mit Constraints/Formeln. Bei der Festlegung was Erreichbarkeit dabei bedeutet, spielt der Bezeichner i , der mit denen der entsprechenden Variablen der Constraints der Form b übereinstimmen muss, eine entscheidende Rolle. Zum Beispiel ist

dann das Label l' vom Label l durch die $\rightarrow_{\Box_i}$ Regel erreichbar, falls im dazugehörigem MCS der Constraint $l <_i l'$ existiert.

- Falls $S = S' \cup \{\Diamond_j \varphi \mid l\}$ und
  1. l' ein neues Label ist, d.h sonst nicht in S vorkommt und
  2. für alle Label l'' mit $l <_j l'' \in S'$ gilt $\varphi \mid l'' \notin S'$,

  dann:

  $$S' \cup \{\Diamond_j \varphi \mid l\} \rightarrow_{\Diamond_j} S' \cup \{\Diamond_j \varphi \mid l \ , \ l <_j l' \ , \ \varphi \mid l'\} \ .$$

- Falls $S = S' \cup \{\Box_i \varphi \mid l\}$ und
  1. $l <_i l^+ \in S'$ sowie $l^- <_i l \in S'$ und
  2. $\varphi \mid l^+ \notin S'$, $\Box_i \varphi \mid l^+ \notin S'$, $\varphi \mid l^- \notin S'$, $\Box_i \varphi \mid l^- \notin S'$ oder $\varphi \mid l \notin S'$

  dann:

  $$S' \cup \{\Box_i \varphi \mid l\} \rightarrow_{\Box}$$
  $$S' \cup \{\Box_i \varphi \mid l \ , \ \varphi \mid l \ \ , \ \Box_i \varphi \mid l^- \ , \ \varphi \mid l^- \ \ , \ \Box_i \varphi \mid l^+ \ , \ \varphi \mid l^+\} \ .$$

Die Abbildungen 3.7 und 3.8 demonstrieren graphisch die neuen modalen Regeln durch strukturierte MCS. Dabei seien Γ und alle Γ_i wieder Formelmengen.

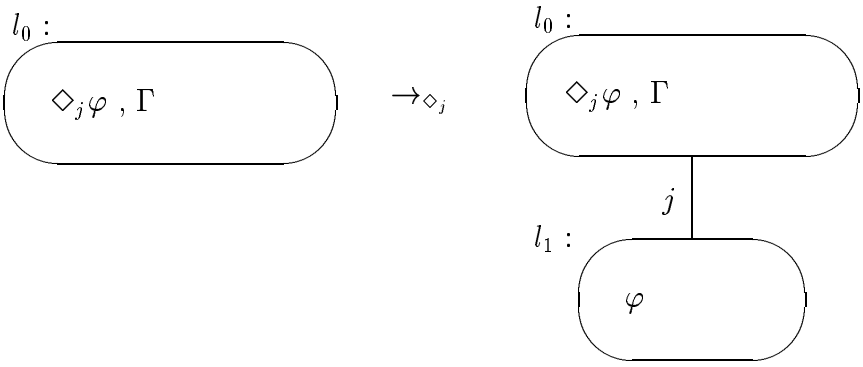

Abbildung 3.7: $\rightarrow_{\Diamond_j}$-Regel

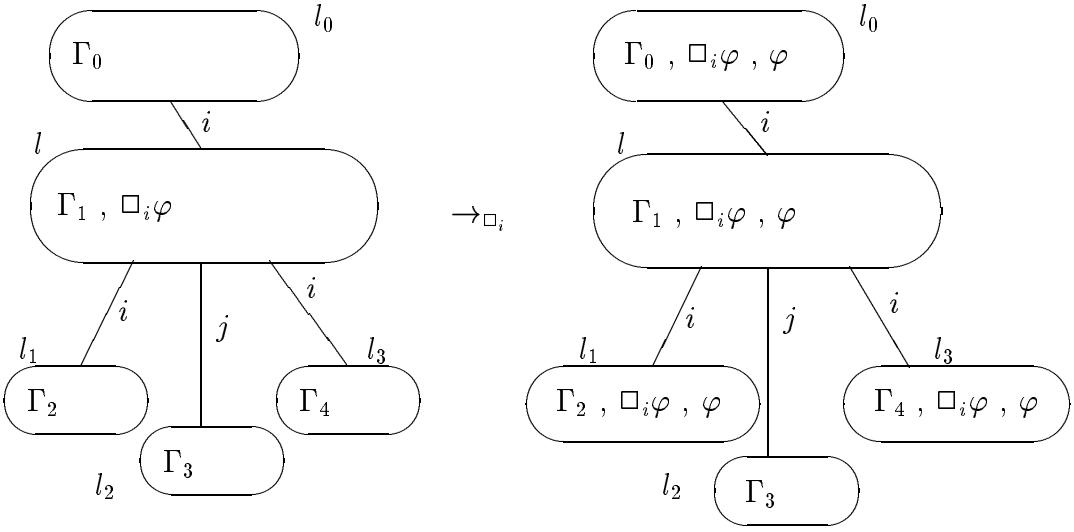

Abbildung 3.8: $\rightarrow_{\Box_i}$-Regel für die Logik $S5_n$

3.2.3 Der Blocking Test

Dieser Test wird wie sein Gegenstück in den vorherigen Tableaus benutzt, d.h. er gibt mittels eines Prädikates an, wann ein Label l ein anderes l' blockiert. Unterschiede treten lediglich in den Bedingungen des Prädikats auf, da auf die Bezeichner der Modaloperatoren Rücksicht genommen werden muss.
$blocked(\Diamond_j\varphi \mid l, S, l')$ gdw

1. S ist l-komplett ,
2. $\varphi \mid l \in S$,
3. es existiert eine natürliche Zahl n mit $n \geq 1$, $l' = l_0$, $l = l_n$ und
   $l_0 <_j l_1 \in S$, $l_1 <_j l_2 \in S$, ... , $l_{n-1} <_j l_n \in S$ sowie
4. $\{\Box_j\psi : \Box_j\psi \mid l \in S\} \subseteq \{\Box_j\psi : \Box_j\psi \mid l' \in S\}$
   d.h. für alle $\Box_j\psi \mid l \in S \Rightarrow \Box_j\psi \mid l' \in S$.

Während des Ablauf des Algorithmus wird der Blocking-Test wieder auf ein ein gerade aktuelles (in Bearbeitung befindliches) Label l_i angewandt.
Getestet wird dabei wieder ob es ein l_k gibt, so dass $blocked(\Diamond_j\varphi \mid l_k, S, l_i)$ gilt.

3.2.4 Das eigentliche Tableauverfahren

Die prinzipielle Vorgehensweise zur Anwendung der Regeln, insbesondere der Reihenfolge, ist gleich der in den anderen Tableaukalkülen. D.h. Ziel des Tableaus ist wieder ein vollständiges, clashfreies sowie ALC-konsistentes MCS herzustellen.
Bei den bisher behandelten Tableaus garantierte dies die Arbeitsweise, für jedes Label l_i von der Wurzel der Baumstruktur der Welten/Label ausgehend, l_i-Vollständigkeit herzustellen. Folglich wird, solange kein Backtracking zum Zug kommt, die Weltenstruktur lediglich einmal durchlaufen. Dies ist nur durch die Form der Regeln dieser Tableaus möglich. Interessant in diesem Zusammenhang sind die modalen Regeln, denn nur sie beeinflussen bei ihrer Anwendung auch andere Label als dasjenige der Formel, auf die sie angewendet werden. Direkt betrifft es aber nur die Box-Regel $\rightarrow_{\Box}$, denn nur diese führt in anderen Welten neue Formeln ein. Die Diamant-Regel $\rightarrow_{\Diamond}$ erweitert nur die bestehende Weltenstruktur, ausschlaggebend ist aber die Art und Weise der Einführung neuer Formeln in bestehende Welten.
Und genau an dieser Stelle tritt der wichtigste Unterschied zwischen den letzten Tableaus und dem neuen auf. Zuvor konnten durch die Box-Regel nur Formeln in anderen Welten eingeführt werden, wenn diese erreichbaren Welten anschaulich gesprochen unterhalb der aktuellen Welt lagen (Nachfolger der Welt sind). Formal heißt das, es werden bei der Anwendung der Regel $\rightarrow_{\Box}$ auf einen Constraint der Form $\Box\varphi \mid l$ neue Constraints $\Box\varphi \mid l'$ und $\varphi \mid l'$ eingeführt, falls der Constraint $l < l'$ vorhanden ist.
Neu ist jetzt, dass zusätzlich noch in den bildlich gesehen oberhalb liegenden Welten (Vorgängerwelten) neue Formeln eingeführt werden. Exakt bedeutet das, es werden bei der Anwendung der Regel $\rightarrow_{\Box_i}$ auf einen Constraint der Form $\Box_i\varphi \mid l$ neue Constraints $\Box_i\varphi \mid l'$, $\varphi \mid l'$, $\Box_i\varphi \mid l''$ und $\varphi \mid l''$ eingeführt, wobei einerseits der Constraint $l < l'$ und andererseits der Constraint $l'' < l$ vorhanden sein muss. Aber diese zusätzlichen neuen Formeln zerstören mit ihrer Einführung in oberhalb gelegene Welten eine schon vorhandene Vollständigkeit bezüglich der entsprechenden Label, denn nach der alten Vorgehensweise wäre das zugehörige MCS schon vollständig bezüglich des Elternlabels der aktuellen Welt.
An der Grafik zur $\rightarrow_{\Box_i}$ Regel [5] erkennt man, vor der Anwendung der Regel war das MCS S l_0-vollständig, danach ist es das nicht mehr, denn es sind zu diesem Zeitpunkt noch nicht alle Formeln mit dem Label l_0 vollständig abgearbeitet worden.

[5].. siehe letzte Grafik im Abschnitt 3.2.2

Die Konsequenz dieser Veränderung ist eine neue Strategie im Ablauf des Tableaus. Startpunkt hierbei ist der Ablauf der alten Vorgehensweise unter Nichtbeachtung der möglicherweise durch die Box-Regel verletzen Vollständigkeiten, also wird die Weltenstruktur wie bei den letzten Tableaus aufgebaut und dabei einmal vollständig durchlaufen. Ist dies abgeschlossen wird bei der Wurzel beginnend die Baumstruktur der einzelnen Labels erneut durchlaufen und dabei für jedes einzelne Label getestet, ob es korrekt entfaltet ist bzw. Vollständigkeit bezüglich dieses Labels besteht. Ist die entsprechende Vollständigkeit gegeben, wird mit dem nächsten erreichbaren Label in gleicher Weise fortgefahren. Falls aber neue Formeln mit dem aktuellen Label durch die Box-Regel hinzugekommen sind, d.h. keine Vollständigkeit bezüglich des Label mehr vorliegt, werden diese Formeln in gleicher Weise, das heißt wie im ersten Teil des Tableaus, entfaltet. Anschliessend wird mit dem nächsten erreichbaren Label der Tableau fortgesetzt.
Dieses Durchlaufen der Baumstruktur mit dem Entfalten zusätzlich hinzugekommener Formeln wird einfach solange wiederholt, bis ein Durchlauf der gesamten Baumstruktur keine neuen Formeln mehr findet. In diesem Sinne kann man vereinfacht den neuen Tableau als mehrfache Wiederholung des alten Tableaus für $S4$ mit erweiterter Box-Regel auffassen.

Im folgendem soll ein kleines Beispiel (Abb. 3.9) die Arbeitsweise graphisch darstellen. Zu beachten ist dabei besonders der letzte Schritt, welcher den soeben erläuterten Unterschied und den damit verbundenen zusätzlichen Durchlauf verdeutlicht.
Zum besseren Verständnis der Arbeitsweise sind im Bild die Pfeile der Regeln in Höhe der Label, auf die sie angewendet wurden, gezeichnet.

Ein weiterer Unterschied des neuen Tableaus besteht in der Bezeichnung der Kanten im Bild bzw. formal in der Differenzierung der modalen Constraints der Form b. Das heißt ab jetzt tragen alle Constraints, die einzelne Label miteinander verbinden, einen zusätzlichen Bezeichner. Dieser muss natürlich bei der Anwendung der Regeln entsprechend beachtet werden. Aber außer dieser zusätzlichen Bedingung ändert sich damit an der Funktionsweise der Regeln nichts.
Bleibt nur noch zu erwähnen, dass der Blocking-Test in angepasster Form [6] sowie der ALC-Test in gleicher Anwendungsweise wie auch

[6] ... muss die neuen Bezeichner der Verbindungsconstraints mit berücksichtigen

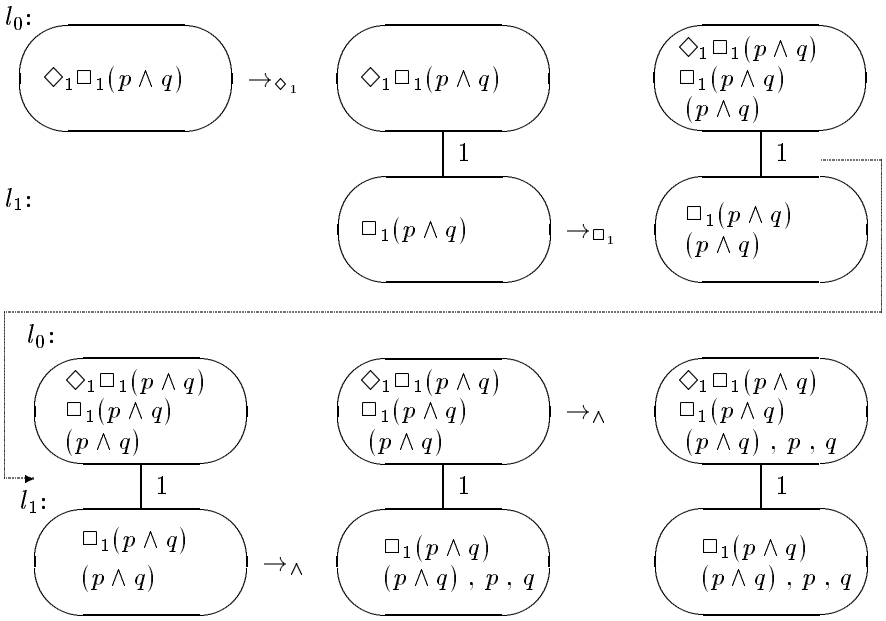

Abbildung 3.9: Beispiel für den Ablauf des polymodalen Tableau

zuvor Teil des Tableaus sind. Insbesondere bei den wiederholten Durchläufen durch die Baumstruktur der Welten muss natürlich, sobald neue Formeln gefunden werden, auch die Konsistenz bezüglich ALC durch den ALC-Test für das entsprechende Label erneut überprüft werden.
Gleiches gilt auch für das Blocking. Falls in einem Label neue Formeln auftauchen ist nach einer vollständigen Entfaltung aller Formeln des betreffenden Labels erneut zu prüfen, ob das Label blockiert werden kann.
Das Backtracking über die verschiedenen Möglichkeiten bei Anwendung der Regel $\rightarrow_\vee$ ist natürlich ebenso wieder Bestandteil des neuen Tableaus. Insbesondere ist es auch im zweiten Teil des Tableaus notwendig, denn durch zusätzliche Formeln können auch ein Clash oder auch ein negatives Ergebnis des ALC-Tests hervorgerufen werden.

Kapitel 4

Ein System nur für Beschreibungslogik

Thema dieser Arbeit ist die Implementierung eines modular aufgebauten Erfüllbarkeitstesters. In Kapitel 1.4 [1] ist das grundlegende Schema der Konstruktion des Testers erläutert worden. In diesem Abschnitt soll nun näher auf den DL-Solver eingegangen werden.

4.1 DL-Solver

Der DL-Solver stellt die Komponente dar, welche die Aufgabe des ALC-Tests aus dem Tableau übernimmt. Formal muss er also die Erfüllbarkeit einer Menge von Gleichungen und Ungleichungen überprüfen. Dies geschieht, indem alle Gleichungen ($C = \top$) als TBox geladen werden, und im Anschluss die Erfüllbarkeit jeder einzelnen Ungleichung ($D \neq \top$) bezüglich der TBox der zugehörigen Gleichungen getestet wird.
Am Ende des ALC-Tests sollten bei positiven Endergebnis alle Ungleichungen erfüllbar bezüglich der TBox sein.
Besteht die zu testende Menge nur aus Gleichungen muss zusätzlich die Konsistenz der geladenen TBox überprüft werden.
Aus dieser Arbeitsweise ergeben sich sofort Anforderungen an einen Kandidaten für einen DL-Solver, das heißt welche grundlegenden Funktionen ein existierendes ALC-System mindestens erfüllen muss, um als DL-Solver einsetzbar zu sein:

- Laden eines ALC-Konzepts als Teil einer TBox,
- Test der Erfüllbarkeit eines ALC-Konzepts bezüglich einer TBox,

[1] siehe Abbildung auf Seite 11

- Aktualisieren der TBox.

Insbesondere beinhaltet der letzte Punkt auch das Löschen einer TBox, denn jeder ALC-Test während Ablauf des Tableaus erzeugt auch wieder eine neue und andere Menge zu testender Gleichungen und Ungleichungen.
Eine weitere Voraussetzung des modularen Aufbaus ist eine praktikable Schnittstelle zwischen dem DL-Solver und dem Rest des Testers. Denn es müssen viele Daten möglichst schnell zwischen den Modulen ausgetauscht werden. Genauer heißt das, die vom ML-Solver an den DL-Solver zu liefernden Daten sollten sehr schnell übertragen werden, damit der DL-Solver seine Antwort, die nur positiv oder negativ ausfällt, generieren kann.

4.2 Verwendung von FaCT

Zur Implementierung des hier vorgestellten Erfüllbarkeitstesters ist als DL-Solver das von Ian R. Horrocks in [1] vorgestellte System FaCT zum Einsatz gekommen. Gründe hierfür waren erstens die Schnelligkeit des Systems auf der ALC-Ebene, zweitens die Verfügbarkeit des Systems in Form einer Client/Server-Anwendung [35] und drittens die erfolgreiche Anwendung dieses Systems für vergleichbare Aufgaben.
FaCT basiert auf einem Algorithmus, der einen Tableaukalkül für ALC und einige Erweiterungen von ALC (z.Bsp. um transitive Rollen) implementiert. Desweiteren erreicht FaCT durch viele Optimierungen sehr gute Leistungen bei Performancevergleichen mit anderen Systemen (siehe hierzu Abb. 7.1 Seite 63). Zum Beispiel sind in FaCT folgende den Tableauablauf betreffende Optimierungen implementiert worden:

- Lazy Unfolding - spätes Entfalten der Negationen, um einen Clash früher entdecken zu können,
- Semantic Branching Search - Vermeidung mehrfacher Abarbeitung von Teilbäumen,
- Dependency Directed Backtracking - clashabhängig gesteuertes Backtracking (sog. Backjumping) und
- Caching - Zwischenspeichern von Teillösungen.

Ausführliche Informationen zu allen genannten Optimierungen befinden sich in Horrocks Arbeit [1].

Das System FaCT ist in Lisp implementiert worden. Eine Kommunikation mit dem laufendem System findet auf einer Konsole statt, das heißt das System liefert einen Prompt, an dem Funktionsaufrufe an das System gestartet werden können. Und diese Form der Kommunikation des Systems mit der Umwelt wird vom direkten Anwender und auch von einem anderen Programm benutzt. Folglich müssen von einem FaCT benutzenden Programm, wie der ML-Solver, Funktionsaufrufe an FaCT in dessen Syntax geschehen. So finden während eines ALC-Testes des Tableaus die folgenden FaCT-Funktionen Anwendung:

1. `(init-tkb)` - Löschen aller definierten Konzepte, Rollen und deren Beziehungen untereinander
2. `(defconcept eq? ...)` - Definition eines Konzepts einer Gleichung
3. `(defconcept andc (and eq? ...))` - Definition eines Konzepts um alle Konzeptgleichungen zusammenzufassen
4. `(implies andc :TOP)` - 4. bis 6. Zusammenfassung als TBox laden und klassifizieren
5. `(implies :TOP andc)` - s.o.
6. `(classify-tkb)` - s.o.
7. `(satisfiable_c (not eq?))` - Test einer Ungleichung, der eine Antwort liefert

Die oben dargestellte Reihenfolge der Funktionsaufrufe entspricht auch der tatsächlich im Programm verwendeten, lediglich die Anzahl der Aufrufe der Form von 2. und 7. könnte grösser als eins sein. Dies hängt natürlich von der Anzahl der zu berücksichtigenden Gleichungen und Ungleichungen ab. Mit dem Aufruf 3. werden alle ALC-Konzepte C_i einer Gleichung $C_i = \top$ in einer Konjunktion zusammengefasst. Diese wird dann mittels 4. , 5. und 6. zur TBox von FaCT erklärt.

Im Anschluss wird mit einem Aufruf der Form 7. für eine jede Ungleichung getestet, ob sie bezüglich der soeben geladenen TBox erfüllbar ist. Diese letzten Funktionsaufrufe liefern Antworten von FaCT. Und falls alle Rückgaben positiv waren, ist auch der ALC-Test positiv beendet worden. Falls hingegen eine Antwort negativ ist, ist sofort auch der gesamte ALC-Test negativ.

Nimmt man in einem Beispiel eine zu testende Welt mit n Gleichungen und m Ungleichung an, dann würde der ALC-Test für diese Welt schon $n + m + 5$ einzelne Funktionsaufrufe von FaCT umfassen.
Im Kapitel 6 über die Kombination der beiden Systeme wird dann noch eine verbesserte Version des ALC-Testes vorgestellt. Diese kommt dann mit weniger Zugriffen aus.

4.3 FaCT-Syntax von ALC-Konzepten

Innerhalb des Abschnitts über die Syntax modaler Beschreibungslogiken ist diese eindeutig definiert worden. Das System FaCT erfordert dafür natürlich seine eigene Schreibweise, diese lehnt sich sehr stark an die Syntax der Programmiersprache Lisp von FaCT an. Deshalb müssen ALC-Konzepte in folgender Form bereitgestellt werden:

`:TOP` für das Zeichen $\top$,

`(and` C_1 ... C_n`)` für $C_1 \sqcap C_2 \sqcap ... \sqcap C_n$,

`(or` C_1 ... C_n`)` für $C_1 \sqcup C_2 \sqcup ... \sqcup C_n$,

`(not` C`)` für $\sim C$

`(some` R C`)` für $\exists R.C$ und

`(all` R C`)` für $\forall R.C$.

Diese Arten der Definition von ALC-Konzepten innerhalb des Systems schöpft nicht alle Möglichkeiten aus. FaCT selber wurde für eine bezüglich ALC ausdrucksstärkere Sprache entwickelt.
Dies ermöglichte dann auch die Verbesserung der Implementierung des ALC-Testes[2].

[2] Genaueres im Abschnitt 6.4

Kapitel 5

Ein System für den Modallogikteil

In diesem Kapitel wird der Hauptteil des Erfüllbarkeitstesters vorgestellt. Dabei handelt es sich um die praktische Umsetzung der im Kapitel 3 beschriebenen Tableaus.
Zuerst werden für den Tableau benötigte Datenstrukturen eingeführt. Danach werden die Umsetzungen der Regeln und deren Zusammenspiel, also der Ablauf der Tableaus, vorgestellt.

5.1 Aufbau verwendeter Datenstrukturen

Basis der eingeführten Tableaus sind modale Constraintsysteme, die einer Menge von gelabelten Formeln und Verbindungsconstraints entsprechen. Strukturierte modale Constraintsysteme ordnen zusätzlich alle Formeln nach ihrem Label und unterteilen dann das gesamte Constraintsystem in einzelne miteinander verbundene Welten. Diese Strukturen stellen auch das Grundgerüst bei der Implementierung dar.
Zur Programmierung wurde die objektorientierte Programmiersprache JAVA benutzt. Deshalb sind die Datenstrukturen in geeigneten Klassen implementiert worden. Zum Einsatz kamen dabei die Klassen:

1. `World` entspricht der Menge von Formeln mit gleichem Label
2. `myWorldList` ist eine Zusammenfassung mehrerer Welten
3. `myNode`, `andNode`, `orNode`, `notNode`, `boxNode`, `diaNode`, `eq_topNode` sind Formelbäume
4. `myNodeList`
   ist eine Zusammenfassung mehrerer Formelbäume.

5.1.1 Speicherung von Formeln

Einzelne Formeln werden in Bäumen gespeichert. Diese Bäume repräsentieren die Syntax einer Formel und sind aus den logischen Junktoren entsprechenden Knoten [1] aufgebaut. Deshalb finden auch die Klassen unter 3. Verwendung.
Die Klassen `andNode` und `orNode` sind dabei mehrstellig ausgelegt, das heißt innerhalb dieser Klassen existiert ein Verweis auf eine Liste von Kindern sogenannten *childnodes*. Diese Liste von Formelbäumen wird durch die Klasse `myNodeList` realisiert. Die anderen Junktorenklassen `notNode`, `boxNode` und `diaNode` verweisen direkt auf einen *childnode*.
Eine besondere Rolle spielt die Klasse `eq_topNode`, die als Umsetzung atomarer Formeln sogenannter Gleichungen eingesetzt wird. Es werden wie schon angekündigt nur Gleichungen der Form $C = \top$ berücksichtigt, wobei C ein ALC-Konzept ist. Folglich stellt die letzte Klasse ein Blatt innerhalb eines Formelbaums dar und beinhaltet das ALC-Konzept der Gleichung.
Ungleichungen sind dann negierte Gleichungen, technisch heißt das ein Objekt vom Typ `notNode` mit dem Verweis seines *childnode* auf ein Objekt vom Typ `eq_topNode`.
Alle Knotenklassen haben zusätzlich eine Nummer um sie innerhalb des Formelbaums eindeutig bestimmen zu können.

5.1.2 Speicherung von MCS

Die zentrale Datenstruktur, um strukturierte MCS im Programm verfügbar zu machen, ist eine Baumstruktur, die aus Objekten der Klasse `World` zusammengesetzt ist. Diese Klasse beinhaltet innerhalb ihres Aufbaus einen Verweis auf eine Liste von erreichbaren Welten, die durch die Klasse `myWorldList` realisiert wird.
Zum Aufbau einer baumartigen Weltenstruktur reicht dieser Verweis, aber um innerhalb dieses Baums navigieren zu können, enthält jede `World`-Klasse noch eine eindeutige Nummer und einen Verweis auf ihren unmittelbaren Vorgänger. Damit kann dann von jeder Welt aus auch rückwärts zur Wurzel des Baums gesprungen werden.
Die Objekte der `World`-Klasse sollen nun innerhalb des Programms Mengen von Formel darstellen. Im Hinblick auf die Regeln des Tableaus gibt es deshalb innerhalb dieser Klasse mehrere Listen von Formeln, realisiert durch die Klasse `myNodeList`. Diese Listen sind:

1. *to_do_formulas*
   speichert alle noch zu bearbeitenden Formeln

[1].. engl. node

2. *dia_nodes*
   speichert alle Formeln mit $\Diamond$ als Hauptjunktor

3. *box_nodes*
   speichert alle Formeln mit $\Box$ als Hauptjunktor

4. *eqtop_and_not_nodes*
   speichert alle Gleichungen und Ungleichungen

Die Trennung der Formeln nach ihren Hauptjunktor wurde dabei hauptsächlich vorgenommen, um den Suchaufwand beim Test ob ein Clash[2] vorliegt so gering wie möglich zu halten. Denn dieser ist Teil fast jeder Anwendung einer Regel des Tableaus. Um nun einen Clash zu entdecken ist es nur noch nötig darauf zu achten, ob bei Einfügen einer Formel in die Liste 4 die neue Formel mit den Elementen dieser Liste keinen Clash bildet.
Die Formeln mit Modaloperatoren als Hauptjunktor werden ebenfalls gesondert innerhalb einer Welt gespeichert, da der Ablauf des Tableaus verlangt, dass keine nichtmodalen Regel mehr vor Anwendung einer modalen Regel angewandt werden kann. Folglich müssen Formeln mit Modaloperatoren zwischengespeichert werden.

5.1.3 Datenstrukturen für den Tableauablauf

Während der Erläuterung der einzelnen Regeln aller Tableaus ist die Regel für die Disjunktion zweier Formeln durch ihre verschiedenen Anwendungsmöglichkeiten aufgefallen. Dies hat zur Folge, dass während des Ablauf des Tableau-Algorithmus, zum Beispiel nach Auftreten eines Clash, ein bestimmter Zustand des MCS wiederhergestellt werden muss.
Diese Zustände müssen im Ablauf des Computerprogramms für den Tableau natürlich irgendwie abgespeichert werden. Die zu rekonstruierenden Zustände hängen durch die Auswahlmöglichkeit direkt mit der Regel $\rightarrow_\vee$ zusammen. Deshalb wird während des Programmablauf bei jeder Anwendung der Regel $\rightarrow_\vee$ im Programm vor dem Ausführen der Regel der Zustand des gesamten MCS, das heißt im Programm die zugehörigen Weltenstruktur, abgespeichert. Gleichzeitig wird der `ODER`-Knoten (`orNode`) dahingehend markiert, dass erkenntlich wird, welche [3] Auswahlmöglichkeit getroffen wurde. Also muss für eine solche Markierung in der Klasse `orNode` ein entsprechendes Feld vorhanden sein.
Nun kommt es im Ablauf des Tableaus auch zur mehrfachen

[2] Siehe Abschnitt 3.1.1

[3] Beachte im Gegensatz zur Regel $\rightarrow_\vee$ ist ein `orNode` mehrstellig, deshalb existiert für jedes Kind des `orNode` eine Auswahlmöglichkeit.

Anwendung der Regel $\rightarrow_\vee$. Deshalb werden die Zustände der Weltenstruktur, die durch die Anwendung der Regel zwischengespeichert werden müssen, in einer als LIFO [4] -Speicher/Stack organisierten Liste abgelegt [5], so dass an der obersten Stelle des Stack der Zustand vor der letzten Anwendung der Regel $\rightarrow_\vee$ steht und immer direkt zur Verfügung steht.
Wird ein solcher Zustand dann rekonstruiert, muss dieser währenddessen natürlich vom Stack *or_ worlds* entfernt werden.

Die Tableaus sind bei ihrer Einführung schon getrennt nach uni- bzw. polymodale Versionen vorgestellt worden. Denn durch den Übergang zur polymodalen Version treten doch erhebliche Unterschiede zwischen den Tableaus auf. Deshalb sind auch zwei Hauptanwendung während der Implementierung entstanden, eine deckt die Tableaus für K, T und $S4$ ab und die andere Anwendung realisiert ausschliesslich $S5_n$.
Unterschiede zwischen den beiden Anwendungen finden sich nicht nur in der die Anwendung der Regeln realisierenden Klasse [6] , sondern auch in denen der zu bearbeitenden Datenstrukturen.
So mussten in der Version für $S5_n$ die Klassen `diaNode` und `boxNode` um ein Feld für den Bezeichner des Modaloperators erweitert werden.
Und dieser Bezeichner muss sich laut Tableau auch an den Kanten zwischen den Welten befinden. Deshalb ist die Klasse `world` ebenfalls um ein solches Feld[7] erweitert worden, welches angibt wie die Kante zu ihrer Elternwelt bezeichnet ist.

Im nächsten Abschnitt wird auf die Umsetzung des Tableaus als Ablauf eines Algorithmus eingegangen und insbesondere auf alle Regelanwendungen. Die Formulierung der einzelnen Regeln ist jedoch sehr formal gehalten und zur Implementierung sind noch weitere Bezeichner innerhalb der komplexen Datenstrukturen einzuführen. Grob gesprochen muss eine jede Komponente innerhalb der Datenstrukturen einen eindeutigen Bezeichner tragen, um durch Regeln bearbeitbar zu sein. In erster Linie handelt es sich bei den Komponenten um einzelne Welten, realisiert durch die Klasse `world`, und auch einzelne Formeln, realisiert durch die Klasse `myNode` als Superklasse von `andNode`, `orNode`, `notNode`, `boxNode`, `diaNode` und `eq_topNode`.

[4] engl. last in first out
[5] Im Programm *or_ worlds* genannt und durch die Klasse `myWorldList` realisiert.
[6] Im Programm `Solver` genannt.
[7] mit *PARENT_INDEX* bezeichnet

5.2 Ablauf des Tableaus

Die zentrale Klasse zur Umsetzung der Tableaus aus dem 3. Kapitel ist die `Solver`-Klasse. Dabei sind zwei verschiedene Versionen entstanden, einerseits die für die unimodalen Logiken K , T sowie $S4$ und andererseits die für polymodales $S5(S5_n)$. Beide Versionen dieser Klasse beinhalten in entsprechenden Feldern die für den Tableau notwendigen Datenstrukturen und Methoden die den Ablauf des Tableaus bzw. die Regeln umsetzen.

5.2.1 Die Implementierung des unimodalen Tableaus

Die Entwicklung der `Solver`-Klasse für die unimodalen Tableaus stand am Anfang der Implementierung des polymodalen Tableau. Denn der konzeptionelle Aufbau dieser Klasse wurde auch für die `Solver`-Klasse des Tableaus für $S5_n$ benutzt.
Zur Beschreibung des Aufbaus dieser Klasse werden die verwendeten Datenstrukturen sowie Methoden der Klasse angeführt. Die wichtigsten Felder dieser Klasse zur Umsetzung der benötigten Datenstrukturen sind:

- *start_world* : Wurzel der Weltenstruktur des Tableaus, vom Typ `World`
- *or_worlds* : Liste von Welten (Typ `myWorldList`), Backtracking-Speicher (Stack)

Innerhalb der `Solver`-Klasse stellt folgende Methode den Ablauf des Tableaus sicher:

- die zentrale Verarbeitungsroutine bzw. Methode für automatische Regelanwendungen:

- *boolean solving(World w)*

Diese Methode greift während ihres Ablaufs auf folgende Hilfsmethoden [8] zu:

- Hilfsmethoden der *solving*-Methode:

- *boolean SAT_test(World w)* : Durchführung des ALC-Tests
- *boolean blocking_world(World w1, World w2)* : Durchführung des Blocking-Tests
- *clash_test_add(World w, myNode f)* : Hinzufügen einer Formel f mit gleichzeitigem Test ob ein Clash ensteht.

[8] Unterprozeduren die nur von der *solving*-Methode benutzt werden.

- *World get_next_world(World w)* : Navigation innerhalb der Weltenstruktur
- *World reco_world()* : Rekonstruktion der Weltenstruktur nach einem Backtrackingschritt
- *add_to_reachable_worlds(World w, myNode f)* : Hinzufügen einer Formel f in allen Nachfolgerwelten dieser Welt w

Die *solving*-Methode selbst wird nur jeweils auf eine Welt angewandt, und muss deshalb innerhalb einer den Ablauf regelnden Prozedur aufgerufen werden. Dies regelt innerhalb der `Solver`-Klasse folgende Prozedur (Procedure 1):

Procedure 1 Ablauf des unimodalen Tableaus

```
1:  // Abarbeitung der gesamten Weltenstruktur
2:  boolean satisfiable = false;
3:  World w = start_world;
4:
5:  while w != null do
6:    satisfiable = solving(w);
7:    if satisfiable then
8:      w = get_next_world(w);
9:    else
10:     w = null;
```

Die Anwendung aller Regeln innerhalb einer Welt wird durch die `solving`-Methode (Procedure 2) realisiert. Diese Methode verzweigt, in Abhängigkeit der gerade in Abarbeitung befindlichen Formel, in dem Formeltyp entsprechende Unterprozeduren. Diese sollten als Makros aufgefasst werden, das heißt diese Quelltextabschnitte stehen im eigentlichen Programm an Stelle ihrer Verweise. Hier werden sie nur der Übersichtlichkeit wegen getrennt dargestellt.
Nachdem alle Einträge der Liste `w.to_do_formulas` durch die äußere `while`-Schleife (Codezeilen 12 bis 26) abgearbeitet worden sind, werden in zwei Schleifen zuerst alle *diaNodes* (Codezeilen 27 bis 32) und dann alle *boxNodes* (Codezeilen 33 bis 38) abgearbeitet. Damit sind dann alle Formeln dieser Welt abgearbeitet. Und es finden zum Abschluss nur noch Blocking-Test (Codezeile 39) und ALC-Test (Codezeile 43) statt.
Wie schon bei der Einführung von Procedure 1 zu sehen war, wird zur Darstellung der Quelltexte ein der Syntax von JAVA ähnlicher Pseudocode benutzt.

Procedure 2 *solving*-Methode

```
1: {Aufruf: boolean solving(World w)}
2: // Abarbeitung aller Formeln einer Welt
3: boolean sat = false;
4: int solving_index = 0;
5: child_index = 0;
6: myNode formula = null;
7: myNode child = null;
8: myNodeList childs = null;
9: World new_world = null;
10:
11: formula = w.to_do_formulas.getmyNode(solving_index);
12: while formula != null do
13:   if (formula instanceof andNode) then
14:     proc_andNode
15:   else if (formula instanceof orNode) then
16:     proc_orNode
17:   else if (formula instanceof notNode) then
18:     proc_notNode
19:   else if (formula instanceof eq_topNode) then
20:     proc_eqtopNode
21:   else if (formula instanceof boxNode) then
22:     proc_boxNode
23:   else if (formula instanceof diaNode) then
24:     proc_diaNode
25:   solving_index = solving_index + 1;
26:   formula = w.to_do_formulas.getmyNode(solving_index);
27: for x = 0 to x < w.dia_nodes.size() do
28:   new_world = new World(w);
29:   formula = w.dia_nodes.getmyNode(x);
30:   child = formula.getallChilds().getmyNode(0);
31:   new_world.to_do_formulas.add(child);
32:   w.reachable_worlds.add_world(new_world);
33: for y = 0 to y < w.box_nodes.size() do
34:   formula = w.box_nodes.getmyNode(y);
35:   child = formula.getallChilds().getmyNode(0);
36:   add_to_reachable_worlds(w,child);
37:   if transitive_flag == true then
38:     add_to_reachable_worlds(w,formula);
39: if blocking_world(w,w.parent_world) then
40:   w.reachable_worlds.clear_world_list();
41:   return sat = true;
42: else
43:   if SAT_test(w) then
44:     return sat = true;
45:   else
46:     reco_macro;
```

Zum Verständnis der gesamten Methode müssen natürlich noch alle Makros [9] hinzugezogen werden. Dabei legen die Makros *proc_andNode*, *proc_orNode*, *proc_eqtopNode* und *proc_notNode* die Behandlung von Formeln mit den klassischen Junktoren fest. Die ersten beiden Makros sind deshalb die direkten Umsetzungen der nichtmodalen Regeln $\rightarrow_\wedge$ und $\rightarrow_\vee$ der Tableaus.
Innerhalb des Makro *proc_andNode* werden einfach alle Kinder des betreffenden `andNode` in Abhängigkeit ihres Formeltyps den verschiedenen Listen der Welt zugeordnet. Dabei kann auch ein Clash enstehen, deshalb ist hier eventuell Backtracking nötig.
Das Makro *proc_orNode* arbeitet grundsätzlich so wie das Makro *proc_andNode*, allerdings wird jeweils nur eines der Kinder ausgewählt und bearbeitet. Die Auswahl hängt stark mit dem Prozess des Backtracking zusammen, denn nach jedem Backtrackingschritt muss als erstes wieder dieses Makro mit der Auswahl des nächsten Kindes des zuletzt bearbeiteten `orNode` ablaufen. Deshalb kann innerhalb dieses Makro ein Clash und desweiteren das Fehlen weiterer Kinder des `orNode` zum Backtracking führen. Ein Backtrackingschritt wird dabei durch das Makro *reco_macro*, welches selbst die Methode *reco_world* aufruft, eingeleitet.
Da sich alle Formeln zur Bearbeitung durch den Tableau in `NNF` befinden müssen, das heißt innerhalb der Formel stehen Negationen nur direkt vor Gleichungen, bearbeiten die letzten beiden Makros Gleichungen und Ungleichungen. Diese werden innerhalb des Programms gleichwertig behandelt[10]. Denn die Methode *clash_test_add* wird mit Ungleichungen oder auch Gleichungen aufgerufen, dannach wird nach einem eventuell enstehenden Clash gesucht und falls vorhanden ein Backtrackingschritt eingeleitet.
Die bisher noch nicht näher erläuterten Makros *proc_boxNode* und *proc_diaNode* kopieren bei ihrer Anwendung die betreffenden Formeln nur in die zugehörigen Listen. Die eigentlichen modalen Formeln werden innerhalb der *solving*-Methode nach Ablauf der äußeren `while`-Schleife (Codezeilen 12 bis 26) in gesonderten `for`-Schleifen bearbeitet. Dabei werden zuerst alle eventuell neu einzuführenden Welten erzeugt. Und im Anschluss Formeln in diese neuen Welten verteilt. Hierbei wird mit Hilfe der globalen Schalter `reflexive_flag` und `transitive_flag` entschieden, welche Formeln verteilt werden müssen.
An dieser Stelle tritt innerhalb der Implementation der Regelanwendungen der einzige Unterschied zwischen den einzelnen Modallogiken auf.

[9] Im Anhang A.1 sind betreffende Makros als Quellen beschrieben.
[10] Deshalb sind beide Makros gleich.

Durch folgende Kombinationen sind die Logiken festgelegt:

K: `reflexive_flag = false` und `transitive_flag = false`

T: `reflexive_flag = true` und `transitive_flag = false`

$S4$: `reflexive_flag = true` und `transitive_flag = true`

Eine besondere Rolle spielt aber der Schalter `reflexive_flag`, denn dieser bewirkt, dass Formeln innerhalb der gerade in Bearbeitung befindlichen Welt hinzugenommen werden. Dies muss aber innerhalb der äußeren `while`-Schleife geschehen, deshalb wird schon innerhalb der Makros *proc_andNode*, *proc_orNode* und *proc_boxNode* auf diesen Schalter Bezug genommen.
Somit kann man nur die erste `for`-Schleife (Codezeilen 27 bis 32) innerhalb der *solving*-Methode als $\rightarrow_\Diamond$-Regel identifizieren. Die Regel $\rightarrow_\Box$ befindet sich dagegen implizit innerhalb der gesamten *solving*-Methode.

Im Anschluss an die *solving*-Methode sollen jetzt noch die *clash_test_add*-, *SAT_test*- und *blocking_world*-Methode vorgestellt werden (Quellen, siehe am Ende des Abschnitts).

Während der Abarbeitung aller Formeln werden alle Gleichungen und Ungleichungen in einer gesonderten Liste (`w.eq_and_not_nodes`) gespeichert. Ungleichungen sind `notNodes` mit einem Kind, das wiederum nur eine Gleichung ist. Eine jede Gleichung bzw. jeder `eq_topNode` ist eindeutig durch eine `ID` gekennzeichnet. Da die *clash_test_add*-Methode mit Gleichungen und auch Ungleichungen aufgerufen werden kann, gibt es auch zwei Möglichkeiten, dass ein Clash enstehen kann. Deshalb wird innerhalb der Methode die Liste `w.eq_and_not_nodes` zweimal durchlaufen und mittels der `ID`s nach einem Clash gesucht. Tritt ein solcher auf, wird die enstprechende Welt markiert (*w.CLASH=true*) und die Suche abgebrochen.

Zur Bearbeitung einer Welt ist nach Vorschrift der Tableaus ein Blockingtest nötig. Dazu wird von der *solving*-Methode die Methode *blocking_world* auf die zu blockierende (gerade in Arbeit befindliche) Welt und deren direkte Vorgängerwelt aufgerufen.
Dabei überprüft die Methode ob für diese zwei Welten die Bedingungen des *blocked*-Prädikats der Tableaus gilt. Rekursiv werden falls nötig weitere Welten getestet.
Liefert diese Methode ein positives Ergebnis, wird innerhalb der

solving-Methode die Liste der eventuell neu enstandenen Nachfolgerwelten gelöscht. Somit wird die weitere Bearbeitung dieses Zweiges der Weltenstruktur blockiert.

Der letzte Schritt zur Abarbeitung einer Welt ist der ALC-Test, der durch die *SAT_test*-Methode implementiert worden ist. Innerhalb des Solvers stellt diese Methode die Schnittstelle zum DL-Solver dar. Diese Verbindung zwischen ML-Solver und DL-Solver wird im nächsten Kapitel noch näher beschrieben.
Innerhalb des Programms wurden zwei verschiedene Möglichkeiten den ALC-Test durchzuführen implementiert. Die Methoden *load_tbox* und *check_SAT* bilden dabei eine Variante. Die andere wird durch die Methode *all_role_solving* realisiert. Ist der Ablauf einer der beiden Möglichkeiten positiv verlaufen, wird diese Welt als erfüllbar (`w.SATISFIABLE = true`) gekennzeichnet. Im Anschluss kann dann mit der nächsten durch die Methode *get_next_world* bestimmten Welt fortgefahren werden.

Procedure 3 *clash_test_add*-Methode

```
 1: {Aufruf: void clash_test_add(World w, myNode node_to_add)}
 2: // Diese Methode wird zur Behandlung von notNodes und eq_topNodes
 3: // benutzt, und innerhalb der Makros der solving-Methode aufgerufen.
 4: myNode child1, child2;
 5:
 6: if node_to_add instanceof notNode then
 7:    {// node_to_add -> notNode}
 8:    child2 = node_to_add.getallChilds().getmyNode(0);
 9:    for index = 0 to (w.eq_and_not_nodes.size()-1) do
10:       child1 = w.eq_and_not_nodes.getmyNode(index);
11:       if (child1 instanceof eq_topNode) then
12:          if (child1.getID()==child2.getID()) then
13:             w.CLASH = true;
14:             break;
15: else
16:    {// node_to_add -> eq_topNode}
17:    for index = 0 to (w.eq_and_not_nodes.size()-1) do
18:       child1 = w.eq_and_not_nodes.getmyNode(index);
19:       if child1 instanceof eq_topNode then
20:          child2 = child1.getallChilds().getmyNode(0);
21:          if node_to_add.getID()==child2.getID() then
22:             w.CLASH = true;
23:             break;
24: if !w.CLASH then
25:    w.eq_and_not_nodes.add(node_to_add);
```

Procedure 4 *blocking_world*-Methode

```
1: {Aufruf: boolean blocking_world(World blocked_w, World blocking_w)}
2: // Diese Methode realisiert den Blocking-Test.
3: // liefert: true falls Blocking-Test positiv (d.h. blocking_w blockiert
   blocked_w), false sonst
4: boolean block = false;
5: boolean exists = true;
6: myNode child = null;
7:
8: if blocking_w == null then
9:    block = false;
10: else
11:    for index = 0 to (blocked_w.box_nodes.size()-1) do
12:       child = blocked_w.box_nodes.getmyNode(index);
13:       exists = exists and blocking_w.box_nodes.exist_Node(child.getID());
14:    for index = 0 to (blocked_w.dia_nodes.size()-1) do
15:       child = blocked_w.dia_nodes.getmyNode(index);
16:       exists = exists and blocking_w.dia_nodes.exist_Node(child.getID());
17:    if exists then
18:       block = true;
19:    else
20:       block = blocking_world(blocked_w, blocking_w.parent_world);
21: return block;
```

Procedure 5 *SAT_test*-Methode

```
1: {Aufruf: boolean SAT_test(World w)}
2: // Diese Methode realisiert den ALC-Test.
3: // liefert: true falls ALC-Test positiv (ALC-Konsistenz), false sonst
4: boolean sat = false;
5:
6: if solve_role_flag then
7:    load_tbox(w);
8:    if check_SAT(w) then
9:       sat = true;
10:      w.SATISFIABLE = true;
11: else
12:    if all_role_solving(w) then
13:       sat = true;
14:       w.SATISFIABLE = true;
15: return sat;
```

5.2.2 Die Implementierung des polymodalen Tableaus

Die Implementierung des polymodalen Tableaus basiert auf einer modifizierten Version der *Solver*-Klasse des letzten Abschnitts. Der Aufbau der Klasse wurde nicht geändert, das heißt innerhalb einer den Ablauf des Tableaus steuerenden Prozedur [11] wird die *solving*-Methode aufgerufen.
Diese Methode ruft während ihres Ablaufs wie zuvor einerseits die unveränderten Methoden *SAT_test()*, *clash_test_add()*, *get_next_world()* und *reco_world()* sowie die modifizierten Methoden *blocking_world()* und *add_to_reachable_worlds()* auf.
Zusätzlich wird zur Steuerung des Tableauablaufs die Methode *check_ready()* benutzt. Diese Methode wird für die Überprüfung aller Welten, ob diese vollständig abgearbeitet worden sind, eingesetzt. Denn der Ablauf des Tableaus [12] verlangt nach Abarbeitung aller Welten den Test, dass auch wirklich alle Formeln in den Welten bearbeitet worden sind.

Procedure 6 Ablauf des polymodalen Tableaus

```
1:  // Abarbeitung der gesamten Weltenstruktur
2:  boolean satisfiable = false;
3:  boolean ready = flase;
4:  World w = null;
5:
6:  while ready==false do
7:    w = start_world;
8:    while  w != null  do
9:      satisfiable = solving(w);
10:     if  satisfiable  then
11:       w = get_next_world(w);
12:     else
13:       w = null;
14:   ready = check_ready(start_world);
```

Somit erklärt sich auch die Notwendigkeit einer zweiten `while`-Schleife (ab Codezeile 6 bis einschliesslich 14), die den bisherigen Ablauf kapselt. Dies hat zur Folge, dass die gesamte Weltenstruktur nicht nur einmal vollständig bearbeitet wird, sondern mehrfach durchlaufen wird um sicherzustellen, dass alle Formeln bearbeitet wurden.
Der Ablauf der *solving*-Methode ist prinzipiell der gleiche, nur auf angepasste Datenstrukturen und entsprechend zusätzliche Bedingungen bei der Umsetztung der Regeln, erforderten die

[11] Diese Prozedur ist in einer extra Methode verankert, die nach dem Erzeugen eines Objekts der *Solver*-Klasse, aufgerufen wird.
[12] Siehe Abschnitt 3.2.4.

Modifizierung [13] der Methoden *add_to_reachable_worlds()* und *blocking_world()* sowie des Teils der *solving*-Methode, der sich um die Einführung neuer Welten kümmert.
Bei letzterem handelt es sich konkret, um den modifizierten Konstruktoraufruf für die Einführung einer neuen Welt, die jetzt zusätzlich noch vom Bezeichner i, der diese Welt erzeugenden Diamanten (Symbol: $\Diamond_i$), abhängt.
Durch diese zusätzliche Abhängigkeit der Welten, werden die benötigten bezeichneten Kanten zwischen den Welten modelliert. Die Erweiterungen der obengenannten Methoden realisieren damit die zusätzlichen Bedingungen:

1. dass bei Ausführung der $\rightarrow_{\Box_i}$- Regel, der Bezeichner i der Box und der Bezeichner der Kante übereinstimmen müssen (innerhalb von *add_to_reachable_worlds()*) und
2. beim Blockingtest die betreffenden Welten, über Kanten mit dem gleichen Bezeichner erreichbar sind (innerhalb von *blocking_world()*).

Alle weiteren Ablaufschritte sind genauso wie im vorherigen Tableau realisierbar, das heißt die entsprechenden Methoden und Makros können für den polymodalen Fall vom unimodalen Tableau übernommen werden. Mit einer Übernahme ist lediglich bei Umsetzung des Backtrackingablaufs auf geeignete Anpassungen für die geänderten Datenstrukturen zu achten.

[13] Siehe im Anhang A.2.

Kapitel 6

Die Kombination von FaCT und SMDL

Innerhalb dieses Kapitels wird die Verbindung der modallogischen und der beschreibungslogischen Komponente näher untersucht. Dabei wird auf die Struktur des gesamten Systems SMDL[1] und auf die notwendige Kommunikation zwischen den Teilen der Struktur eingegangen. Anschliessend werden Optimierungen, die im direkten Zusammenhang mit der modularen Struktur stehen, vorgestellt.

6.1 Modulare Aufbauweise

Innerhalb der Einleitung, genauer im Abschnitt 1.4, ist schon die Grobstruktur des Aufbaus unseres Beweisers dargestellt worden. Diese Darstellung soll in diesem Abschnitt um eine weitere Komponente bzw. Modul erweitert werden. Denn die erste Darstellung konzentrierte sich nur auf das Zusammenspiel der Module während des Ablauf des Tableaus, wobei dort ein Modul den modallogischen Anteil verarbeitet und zwischendurch auf das zweite Modul, das den ALC-Anteil bearbeitet, zugreift. Für die Implementierung des Beweisers ist es praktikabel diese Module als ein Client/Server-System aufzufassen, mit dem ML-Solver als Client, der während seines Programmablaufs auf den FaCT-Server bzw. DL-Solver zugreift. Abbildung 6.1 zeigt die erweiterte Client/Server-Struktur.

Eine wichtige Vorbedingung des Tableaus ist aber die Gestalt der Formel, deren Erfüllbarkeit überprüftwerden soll, in NNF [2].
Da die zu bearbeitende Formel sowieso in die vom ML-Solver

[1] ... Solver for Modal Description Logics
[2] ... Negationsnormalform, das heißt Negationen treten nur vor Gleichungen auf

benutzte Datenstruktur eingelesen werden muss, wurde dieser Arbeitsschritt noch um die Umwandlung der Formel in NNF-Gestalt und eine Optimierung der entstehenden Datenstruktur erweitert. Und diese drei Verarbeitungschritte wurden in einem Vorverarbeitungsmodul zusammengefasst. Dieses zusätzliche Modul kann auf Wunsch zur Optimierung der Datenstruktur, das heißt zum Erkennen äquivalenter ALC-Konzepte, auch auf den FaCT-Server zugreifen. Dies reduziert eventuell die Anzahl der im Tableau auftretenden Gleichungen. Damit kann aber die Leistungsfähigkeit des Beweisers sowohl positiv als auch negativ beeinflusst werden. Von Vorteil ist eine geringere Anzahl von Gleichungen bzw. unterschiedlichen ALC-Konzepten, da damit weniger Daten bearbeitet werden müssten. Der für die notwendigen Anfragen an den FaCT-Server entstehende Mehraufwand kann einen möglichen Vorteil aber auch zunichte machen.
Desweiteren wurde zum Einlesen der Formel ein externer in JAVA geschriebener XML-Parser [33] [3] benutzt. Dieser Parser ist sehr leicht in ein Programm zu integrieren, weil er in einem abgeschlossenen API mit sehr guter Dokumentation zugänglich ist und damit nur eine Erweiterung der Programmiersprache JAVA darstellt.
Die Definition der Syntax der modalen Beschreibungslogiken ist innerhalb von XML mittels geeigneter `dtd`-Dateien relativ leicht möglich. Siehe dazu im Anhang B die `dtd`-Datei für $S5_n$.
Ein weiteren Vorteil bietet der Parser, indem dieser zu überprüfende Daten in eine interne baumartige Datenstruktur übernimmt, die auf einem Standard [4] der W3C [5] aufbaut, und selbst sehr leicht in die Datenstruktur des Tableau kopiert werden kann.
Desweiteren erlaubt die automatische Erzeugung der NNF-Gestalt einer Formel, auch die Zulassung von Implikationen [6] zur Modellierung von Formeln, denn diese können während der Umwandlung in NNF beseitigt ($\varphi \rightarrow \psi$ wird zu $\neg\varphi \vee \psi$) werden.

[3] Teil des *Simple API for XML(SAX)*.
[4] *Document Object Model (DOM)*, entsprechende Klassen innerhalb von *SAX* vorhanden
[5] Standardisierungsorganisation des Internet, ist u.a. für XML zuständig.
[6] im Anhang B: `ELEMENT IMP` ... der *dtd*-Datei

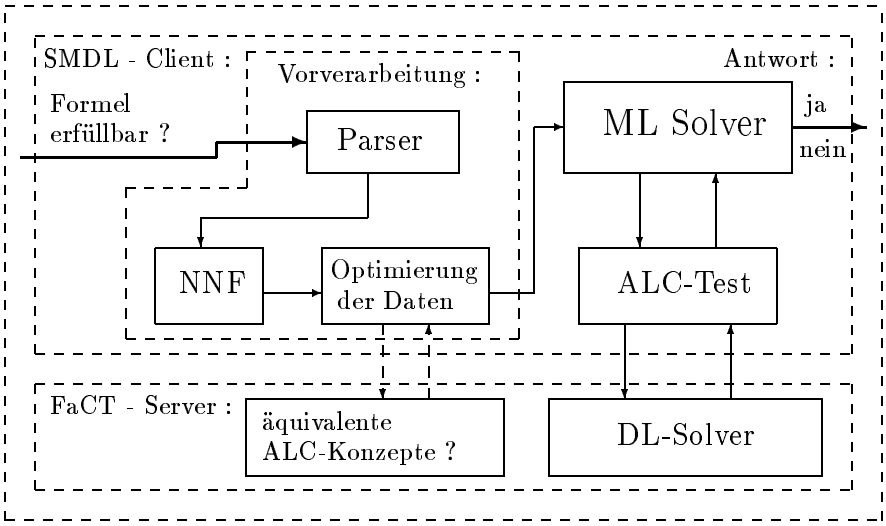

Abbildung 6.1: Modularer Aufbau des Beweisers

6.2 Kommunikation zwischen FaCT und SMDL

Der modulare Aufbau des Systems SMDL war das zentrale Ziel der Entwicklung, denn es sollte mit Hilfe existierender Beschreibungslogiksysteme ein Beweiser für eine ausdrucksstärkere Sprache geschaffen werden. Der Teil des Systems, der für die Beschreibungslogik zuständig ist, sollte austauschbar sein. Dies erfordert zumindest eine modulare Struktur des Aufbaus des neuen Beweisers. Die Modularität kann aber unterschiedlich stark ausgeprägt sein. Um eine größtmögliche Trennung der einzelnen Teile zu erreichen, ist das System, wie oben erläutert, in eine Client/Server-Architektur gebracht worden. Die Kommunikation zwischen SMDL-Client und FaCT-Server beruht auf Sockets. Ein Socket ist eine Schnittstelle zur Datenübertragung zwischen zwei Prozessen. Näheres dazu in [34].
Es ist während der Entwicklung auch eine CORBA-Schnittstelle, wie sie die Entwickler von FaCT in ihrer Client/Server-Architektur verwenden (siehe [35]), erprobt worden. Diese stellte sich innerhalb unseres Projektes als zu langsam heraus, und ist deshalb durch die rudimentäre aber schnellere Socketschnittstelle ersetzt worden.
Durch Verwendung dieser, aus der Netzwerkprogrammierung bekannten Schnittstelle, ist es möglich den SMDL-Client sowie den FaCT-Server auf verschiedenen Rechnern ablaufen zu lassen. Damit ergibt sich natürlich eine sehr grosse Unabhängigkeit zwischen den einzelnen Modulen, und man kann von einer sehr konsequent umgesetzten Modularität sprechen. Dies wird zusätzlich noch durch

die Plattformunabhängikeit der Programmiersprache JAVA unterstützt, wobei es zum Beispiel möglich ist den SMDL-Clienten auf einem Rechner mit anderem Betriebssystem, als das auf dem der FaCT-Server läuft, ablaufen zu lassen.

6.3 Optimierungen der Kombination

Bezüglich des modularen Aufbaus des Beweisers ist die Kommunikation zwischen Client und Server der wichtigste Ansatzpunkt für Optimierungen. Dabei kann der Datenaustausch qualitativ und quantitativ verändert werden. Qualitativ heißt hierbei eine Veränderung der Art und Weise der Anfrage an den FaCT-Server, und quantitativ betrifft natürlich eine Veränderung des Umfangs der einzelnen Datenübertragung sowie der Anzahl aller Anfragen.

6.3.1 Optimierte Datenstrukturen

Formeln sind innerhalb der Datenstrukturen für den Tableau zwar in Baumstrukturen gespeichert, die implementierten Regeln arbeiten aber mit Listen, die solche Baumstrkturen beinhalten. Alle diese Listen müssen in der Regel vollständig abgearbeitet werden. Deshalb führt eine Verkürzung dieser Listen im allgemeinen zu einer besseren Performance.
Werden zum Beispiel während des Ablaufs des Vorverarbeitungsmoduls äquivalente ALC-Konzepte entdeckt, können diese untereinander identifiziert werden. Und nur noch jeweils eins dieser Konzepte in Form einer Gleichung oder Ungleichung muss abgespeichert werden.
Dies ermöglicht auch das bessere Entdecken eines vorhandenen Clash, der sonst eventuell erst während des ALC-Tests entdeckt worden wäre. Somit wird nicht nur das Datenvolumen verringert, sondern auch potenziell weniger zeitintensiver Datenaustausch während Ablauf des Tableaus erreicht.

Schlechtes Laufzeitverhalten insbesondere bei Bearbeitung großer Formeln der Benchmarks des nächsten Kapitels ist sicherlich auf das notwendige Backtracking innerhalb des Tableauablaufs zurückzuführen. Mit der Erläuterung der Datenstrukturen [7] ist schon die Verwendung mehrstelliger *ODER*- und *AND*-Knoten innerhalb der Formelbäume angedeutet worden. Insbesondere jeder *ODER*-Knoten ruft einen möglichen Backtrackingschritt hervor.

[7] siehe Abschnitt 5.1ff

Deshalb werden während der Vorverarbeitung geschachtelte *ODER*- und auch *AND*-Knoten beseitig.

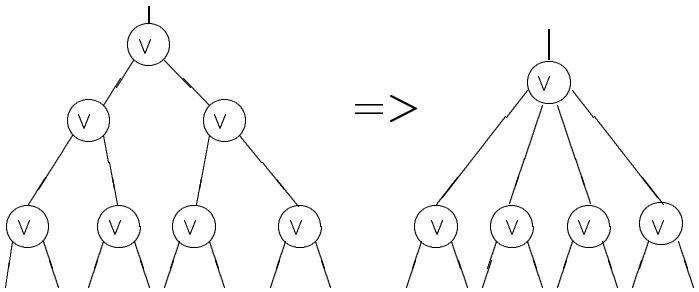

Abbildung 6.2: Beseitigung geschachtelter ODER-Knoten

Einerseits wird durch solch einen Bearbeitungsschritt (Abb. 6.2) die `to_do_formulas`-Liste um ein Element verkürzt, und andererseits ein möglicher Rückpunktschritt beim Backtracking eingespart. Letzteres verringert die abzuspeichernde Datenmenge erheblich, da mit jedem *ODER*-Schritt die gesamte Weltenstruktur zwischengespeichert werden muß.

6.3.2 Eine modifizierte Anfrageform

Innerhalb des Abschnitt 4.2 ist schon die Form der Funktionsaufrufe während einer Anfrage an FaCT erläutert worden. Dabei handelte es sich um das grundlegende Schema des Ladens einer TBox in FaCT, und der Anfrage nach der Erfüllbarkeit eines Konzeptes bezüglich der soeben geladenen TBox. Dies erfordert eine ganze Reihe sequentiell abzuarbeitender Funktionsaufrufe an FaCT.
Die Kommunikation über die Socketschnittstelle erfordert aber bei jedem Senden eines Funktionsaufruf auch das Empfangen der eventuell leeren Antwort. Deshalb ist es sinnvoll die Anzahl der benötigten Funktionsaufrufe so klein wie möglich zu halten. Hierzu wurde die Umsetzung der Anfrage an FaCT modifiziert. Basierend auf den theoretischen Betrachtungen des nächsten Abschnitts (Beweis für All-Role-Solving) wurde die Anfrage dahingehend modifiziert, dass für jede Ungleichung nur noch je ein Funktionsaufruf nötig wird. Das von der Anzahl der Gleichungen abhängige Laden der TBox entfällt, und erspart somit die jeweilig notwendigen Funktionsaufrufe ([8]).
In der Praxis zeigten sich damit auch deutliche Verbesserungen in der benötigten Bearbeitungszeit für verschiedene Formeln. Die folgende Tabelle zeigt die erzielten Verbesserungen anhand einiger

[8] vgl. Abschnitt 4.2

Formeln, die Bestandteil eines Benchmarks[9] für die Logik K sind. Die starken Schwankungen der Faktoren ist dabei sicherlich auf die unterschiedlichen Formelstrukturen zurückzuführen, die Anzahl und Datenumfang der Anfragen der einzelnen ALC-Tests beeinflussen.

Formel	Zeit [sec.]		Faktor
	normal	All-Rolle	besser
k-branch-p(1)	2,8	0,3	9,3
k-branch-p(2)	346	72	4,8
k-t4p-n(2)	27	2,1	12,8
k-t4p-n(3)	126	17,4	7,3
k-d4-p(2)	7,3	0,5	14,6
k-d4-p(3)	53	6,2	8,6

Abbildung 6.3: Bearbeitungszeiten einiger Formeln mit und ohne optimierter Anfragetechnik

6.4 Beweis für All-Role-Solving

Dieser Abschnitt liefert die theoretische Begründung für die Möglichkeit, die Anfrage an FaCT während des ALC-Tests zu modifizieren.
Seien unmittelbar vor Ausführung eines ALC-Testes C_1, ... , C_n die Gleichungen des betreffenden Labels und D_1, ... , D_m die Ungleichungen. Dann würden normalerweise alle Gleichungen als TBox von FaCT geladen, und anchließend jede Ungleichung bezüglich dieser TBox auf Erfüllbarkeit hin getestet.
Die neu entwickelte Anfrageform hingegen testet lediglich für jede Ungleichung D_i die Erfüllbarkeit des folgenden Konzepts

$$C_1 \wedge ... \wedge C_n \wedge D_i \wedge (\forall R.(C_1 \wedge ... \wedge C_n)) \ ,$$

wobei R transitiv ist und jede andere Rolle, die in $C_1, ..., C_n, D_1, ..., D_m$ vorkommt, Subrolle von R ist.
Das Laden einer TBox kann somit entfallen.
Das folgende Lemma wird für den Beweis des sich anschließenden Satzes benötigt. Dieser zeigt, dass die neue Anfrageform wie auch die alte die ALC-Konsistenz eines Labels gewährleistet, genauer die Existenz eines entsprechenden ALC-Modells zeigt.

Lemma 1 *Sei* $x \in \Delta$ *und seien* $I = \langle \Delta, C_1^I, ..., C_n^I, R_1^I, ..., R_m^I \rangle$ *sowie* $I^x = \langle \Delta^x, C_1^{I^x}, ..., C_n^{I^x}, R_1^{I^x}, ..., R_m^{I^x} \rangle$ *ALC-Modelle mit folgenden*

[9] vgl. dazu nächstes Kapitel über Tests und Benchmarks

Eigenschaften :

$$\Delta^x = \{x\} \cup \{y \mid x(R_1^I \cup ... \cup R_m^I)^* y\} \tag{6.1}$$

$$C_i^{I^x} = C_i^I \cap \Delta^x \qquad i \in \{1...n\} \tag{6.2}$$

$$R_j^{I^x} = R_j^I \cap \Delta^x \times \Delta^x \qquad j \in \{1...m\} \tag{6.3}$$

Dann gilt für alle $y \in \Delta^x$ und für alle Konzepte C :

$$C^I \cap \Delta^x = C^{I^x} \cap \Delta^x. \tag{6.4}$$

Beweis. Induktion über den Formelaufbau von C:

- `Induktionsanfang`
  Sei C eine atomare Formel. Dann gilt $y \in C^I \Leftrightarrow y \in C^{I^x}$ nach Definition von I^x .

- `Induktionsvoraussetzung`

$$y \in C^I \Leftrightarrow y \in C^{I^x} \tag{6.5}$$

- `Induktionsschritt`

  1. C sei eine Konjunktion von zwei ALC-Konzepten C_1 und C_2 .

$$\begin{aligned} y \in (C_1 \sqcap C_2)^I &\Leftrightarrow y \in C_1^I \cap C_2^I \\ &\Leftrightarrow y \in C_1^I \ \textit{und} \ y \in C_2^I \end{aligned}$$

  (wegen Induktionsvoraussetzung)

$$\begin{aligned} &\Leftrightarrow y \in C_1^{I^x} \ \textit{und} \ y \in C_2^{I^x} \\ &\Leftrightarrow y \in C_1^{I^x} \cap C_2^{I^x} \\ &\Leftrightarrow y \in (C_1 \sqcap C_2)^{I^x} \end{aligned}$$

  2. C sei Negation des ALC-Konzepts C_1 .

$$y \in (\sim C_1)^I \Leftrightarrow y \in (\Delta \backslash C_1^I)$$

  (wegen $y \in \Delta^x$)

$$\begin{aligned} &\Leftrightarrow y \in (\Delta^x \backslash C_1^I) \\ &\Leftrightarrow y \in \Delta^x \ \textit{und} \ y \notin C_1^I \end{aligned}$$

  (wegen Induktionsvoraussetzung)

$$\begin{aligned} &\Leftrightarrow y \in \Delta^x \ \textit{und} \ y \notin C_1^{I^x} \\ &\Leftrightarrow y \in (\Delta^x \backslash C_1^{I^x}) \\ &\Leftrightarrow y \in (\sim C_1)^{I^x} \end{aligned}$$

3. C habe die Form $(\exists R.C_1)$, wobei C_1 ALC-Konzept sei.

$$y \in (\exists R.C_1)^I \Leftrightarrow y \in \{q \in \Delta \mid \exists z(qRz \wedge z \in C_1^I)\}$$

(wegen $y \in \Delta^x$)

$$\Leftrightarrow y \in \{q \in \Delta^x \mid \exists z(qRz \wedge z \in C_1^I)\}$$

(wegen Induktionsvoraussetzung)

$$\Leftrightarrow y \in \{q \in \Delta^x \mid \exists z(qRz \wedge z \in C_1^{I^x})\}$$

$$\Leftrightarrow y \in (\exists R.C_1)^{I^x}$$

□

Satz 1 *Sei die Rolle R_0 nicht in den ALC-Konzepten C und D enthalten. Dann sind folgende Bedingungen äquivalent:*

(1) Das Konzept $\forall R^0.C \sqcap C \sqcap \sim D$ ist in einem ALC-Modell $I := \langle \Delta, C_1^I, ..., C_n^I, R_1^I, ..., R_m^I, R^{0^I} \rangle$ erfüllbar, mit

- *R^{0^I} ist transitiv und*
- *für alle R_i aus C und alle R_j aus D gilt: $R_i^I \subseteq R^{0^I}$ bzw. $R_j^I \subseteq R^{0^I}$.*

(2) Die Menge $\{C = \top, D \neq \top$[10]$\}$ ist erfüllbar.

Beweis :

- Sei $I = \langle \Delta, C_1^I, ..., C_n^I, R_1^I, ..., R_m^I \rangle$ ein ALC-Modell für $C = \top$ und $D \neq \top$. Dann konstruiere man aus I folgendes ALC-Modell I' :
  $I' = \langle \Delta', C_1^{I'}, ..., C_n^{I'}, R_1^{I'}, ..., R_m^{I'}, R^{0^{I'}} \rangle$, wobei
  - $\Delta' = \Delta$,
  - $C_i^{I'} = C_i^I$ für $i \in \{1, ..., n\}$,
  - $R_j^{I'} = R_j^I$ für $j \in \{1, ..., m\}$ und
  - $R^{0^{I'}} = \Delta \times \Delta$.

  I' ist nach Konstruktion ebenfalls ein ALC-Modell für $C = \top$ und $D \neq \top$. Folglich existiert ein $x \in \Delta'$ mit
  1. $x \in (\sim D)^{I'}$, da $(\sim D)^{I'} \neq \emptyset$ und $D^I = D^{I'}$,
  2. $x \in C^{I'}$, da $C^I = C^{I'}$ und
  3. $x \in (\forall R^0.C)^{I'}$, da $C^{I'} = C^I = \Delta$.

[10] entspricht $\neg(D = \top)$

Also gilt auch $x \in (\forall R^0.C \sqcap C\sqcap \sim D)^{I'}$, wobei R^0 durch die Konstruktion ${R^0}^{I'} = \Delta \times \Delta$ die geforderten Bedingungen trivialerweise erfüllt. Das heißt I' ist auch ein Modell für das Konzept $\forall R^0.C \sqcap C\sqcap \sim D$.

- Sei nun die Formel $\forall R^0.C \sqcap C\sqcap \sim D$ in einem ALC-Modell I erfüllbar, mit $I = \langle \Delta, C_1^I, ..., C_n^I, R_1^I, ..., R_m^I, {R^0}^I \rangle$. Dann existiert ein $x \in \Delta$ mit $x \in (\forall R^0.C \sqcap C\sqcap \sim D)^I$. Durch erneute Konstruktion eines neuen ALC-Modells I' aus I mit folgenden Eigenschaften

  - $\Delta' = \{x\} \cup \{y \mid x(R_1^I \cup ... \cup R_m^I \cup {R^0}^I)^* y\}$,
  - $C_i^{I'} = C_i^I \cap \Delta' \qquad i \in \{1, ..., n\}$,
  - $R_j^{I'} = R_j^I \cap \Delta' \qquad j \in \{1, ..., m\}$ und
  - ${R^0}^{I'} = {R^0}^I \cap \Delta'$,

  ist, nach Lemma 1 und unter Berücksichtigung der Definition von R^0 , I' ein Modell für die Formeln $C = \top$ und $D \neq \top$.

□

Kapitel 7

Tests und Benchmarks

7.1 Benchmarkmethode

Für die klassischen Modallogiken K, KT und $S4$ existieren verschiedenste Beweiser. Da aber zwischen Beschreibungslogiken und Modallogiken enge Verbindungen (siehe [3]) bestehen, werden oft Beschreibungslogiksysteme als Beweiser für die obengenannten Modallogiken herangezogen.
Die neuere Entwicklung von Beschreibungslogiksystemen verfolgt oft neue den Tableauablauf optimiernde Ansätze (siehe [4]).
Entwickler von Modallogikbeweisern haben diese Ansätze übernommen bzw. haben oft Beschreibungslogiksysteme für die entsprechenden Modallogiken angepaßt. Deshalb findet hier ein Vergleich des Systems SMDL mit anderen Beweisern, die in erster Linie für Beschreibungslogiken entwickelt worden sind, statt.
Der Beweiser dieser Arbeit (SMDL) bearbeitet modale Beschreibungslogiken. Die nachfolgenden Testergebnisse vergleichen aber nur die Performance bezüglich der modallogischen Sprachkomponente, das heißt es wurden nur Formeln mit primitiven ALC-Konzepten benutzt. Für die Tests wurde die von Balsiger, Heuerding und Schwendimann in [5] vorgeschlagene Methode umgesetzt. Diese stellt für die Modallogiken K, KT und $S4$ jeweils neun Klassen von beweisbaren[1] und nicht[2] beweisbaren Formeln zur Verfügung. Jede Formelklasse wird durch eine Formel charakterisiert, deren Aufbau von einem Parameter p, mit $1 \leq p \leq 21$, abhängig ist. Mit grösser werdendem Parameter wächst der Aufbau einer Formeln, ebenso wie die Schwierigkeit die Formel beweisen zu können. Die Ergebnisse der Testmethode setzen sich aus den größten Parametern einer jeden Formelklasse zusammen, die innerhalb von 100 Sekunden erfolgreich

[1] ...engl. provable, Tabelle p-Spalten
[2] ...engl. non-provable, Tabelle n-Spalten

bearbeitet werden konnten.

Die folgende Benchmarktabelle zeigt die erreichten Resultate für SMDL im Vergleich zu den Systemen FaCT[1], DLP[30], KSAT[31] und Kris[17]. FaCT und dessen Weiterentwicklung DLP sind optimierte Systeme für Beschreibungslogik, KSAT ist ein optimierter Modallogikbeweiser und Kris ein nicht optimiertes Beschreibungslogiksystem.

	FaCT		DLP		KSAT		Kris		SMDL	
K	p	n	p	n	p	n	p	n	p	n
branch	6	4	19	13	8	8	3	3	2	6
d4	>20	8	>20	>20	8	5	8	6	4	3
dum	>20	>20	>20	>20	11	>20	15	>20	4	11
grz	>20	>20	>20	>20	17	>20	13	>20	4	>20
lin	>20	>20	>20	>20	>20	3	6	9	4	3
path	7	6	>20	>20	4	8	3	11	2	5
ph	6	7	7	9	5	5	4	5	2	3
t4p	>20	>20	>20	>20	10	18	7	5	2	3
KT	p	n	p	n	p	n	p	n	p	n
45	>20	>20	>20	>20	5	5	4	3	1	1
branch	6	4	19	12	8	7	3	3	2	6
dum	11	>20	>20	>20	7	12	3	14	0	3
grz	>20	>20	>20	>20	9	>20	0	5	0	>20
md	4	5	3	>20	2	4	3	4	4	5
path	5	3	16	14	2	5	1	13	1	3
ph	6	7	7	>20	4	5	3	3	2	3
t4p	4	2	>20	>20	1	1	1	7	1	3
S4	p	n	p	n	p	n	p	n	p	n
45	>20	>20	>20	>20	-	-	-	-	1	0
branch	4	4	18	12	-	-	-	-	1	6
grz	2	>20	>20	>20	-	-	-	-	0	18
ipc	5	4	10	>20	-	-	-	-	2	3
md	8	4	3	>20	-	-	-	-	6	4
path	2	1	15	15	-	-	-	-	1	3
ph	5	4	7	>20	-	-	-	-	2	3
t4p	5	3	>20	>20	-	-	-	-	1	0

Abbildung 7.1: Ergebnisse für die Logiken K, KT und $S4$

Die Testergebnisse für die anderen Systeme sind aus [4] übernommen worden. Da die Systeme KSAT und Kris nicht in der Lage sind Formeln für die Logik $S4$ zu beweisen, sind in der Tabelle dafür keine Einträge vorhanden.

Technologisch betrachtet stehen SMDL und Kris auf dem selben Entwicklungstand, beides sind unoptimierte Systeme. Dementspre-

chend erreicht SMDL für die Modallogiken K und T auch nur Resultate, die sich auf dem Niveau der Ergebnisse von Kris bewegen. Auch für die Logik $S4$ erreicht SMDL nicht die Performance des optimierten Systems FaCT.
Allerdings spielte bei der Entwicklung von SMDL dessen modularer Aufbau die wichtigste Rolle. Und trotz dieses Nachteils für die Benchmarkmessungen haben die Tests angemessene Performancewerte geliefert.

7.2 Testumgebung

Die Tests von SMDL sind auf einem PC, Athlon CPU mit 700 MHz und Arbeitsspeicher von 128 MB, unter Linux durchgeführt worden. Zur JAVA-Programmierung wurde das JDK1.2.2RC4 für Linux von den Blackdown[36] Entwicklern benutzt. FaCT wurde in einer angepaßten[3] Client/Server-Version benutzt [35].
Die Formeln der oben beschriebenen Benchmarkmethode wurden der Syntax der modalen Beschreibungslogiken angepaßt, indem Atome durch primitive ALC-Konzepte ersetzt wurden. Diese Syntax unterstützt keine Äquivalenzen, deshalb konnte pro Logik eine Formelklasse nicht berücksichtigt werden[4].
Während Ablaufs der Tests fand die notwendige Kommunikation zum FaCT-Server statt, der selbst lokal auf dem selben Rechner ablief.
Gerade der Aspekt der zusätzlich nötigen Kommunikation während der Tests stellt einen technischen Nachteil gegenüber den anderen Beweisern im Test dar. Dieser konnte aber durch den Einsatz besserer Rechentechnik[5] wettgemacht werden. Denn die Tests lieferten Ergebnisse, die insgesamt auf dem Niveau der Resultate des ebenfalls unoptimierten Systems Kris lagen. An einigen Stellen konnten sogar bessere Werte als FaCT erreicht werden.

[3] FaCT benötigt Allegro-Lisp[37], welches teilweise in der C/S-Version enthalten ist.
[4] Deshalb sind pro Logik nur 8 Zeilen in der Tabelle vorhanden.
[5] Die Ergebnisse der anderen haben ein gewisses Alter, sind deshalb auf potenziell langsamerer Rechentechnik durchgeführt worden.

Kapitel 8

Fazit und Ausblick

8.1 Gegenwärtiger Entwicklungsstand

Der Beweiser SMDL in der Version für die Logiken K, KT und $S4$ ist hinsichtlich der modallogischen Sprachkomponente ausführlich getestet worden. Allerdings sind die Testformeln für die durchgeführten Tests konstruiert worden, und teilweise künstlich verkompliziert worden, um die Bearbeitungsschwierigkeit zu steigern. Deshalb sind diese Tests nicht unbedingt praxisrelevant. Für die polymodale Version mangelte es für ausführliche Tests an geeigneten Beispielen.
Für aussagekräftigere Tests von SMDL sind für geeignete Testbeispiele natürlich nicht nur primitive/triviale ALC Konzepte in den Formeln notwendig. Mit geeigneten Formeln unter den obengenannten Testmodalitäten ist sicherlich das Verhältnis der Bearbeitungszeiten aller Module untereinander von Interesse. Im Abschnitt 6.3.2 ist eine optimierte Anfrageform eingeführt worden. Diese bewirkte insgesamt bessere Gesamtverarbeitungszeiten[1]. Durch Beobachtung der Verteilung der Systemlast[2] konnte eine Verschiebung der Hauptrechenlast vom FaCT-Server zum SMDL-Client beobachtet werden, was Vorraussetzung für weitere Optimierungen im Bereich des Tableauablaufs ist.
Das letzte Beispiel gibt einen kurzen Anreiz, welche Tests insbesondere für den modularen Aufbau notwendig wären. Dabei ist der Einsatz des Systems unter zwei Rechnern noch nicht berücksichtigt. Letzteres ist in LAN [3]-Umgebungen sogar erfolgreich getestet worden, das heißt der FaCT-Server lief auf einem anderen Rechner als der SMDL-Client. Diese Verteilung der notwendigen Rechenlasten auf zwei Rechner kommt natürlich der Gesamtrechenzeit zugute,

[1] ...siehe Abb. 6.3 Seite 58
[2] Unter Linux mit Hilfe des Programms `top` möglich.
[3] ...engl. Local Area Network

und der modulare Aufbau des Systems kann all seine Vorteile ausspielen.
Insgesamt kann damit der Versuch, einen modular aufgebauten Beweiser für modale Beschreibungslogiken zu konstruieren, als gelungen betrachtet werden. Es mangelt lediglich an Beispielen, die die angebotene Modellierungssprache ($S5_n$) benutzten. Aber, wie in den ersten Kapiteln angedeutet, stehen die Chancen, im Umfeld von Multiagentensystemen geeignete Anwendungen zu finden, recht gut.

8.2 Weitere Entwicklungsmöglichkeiten

Für die Zukunft bietet das Konzept eines modular aufgebauten Beweisers, genauer eines modularen Erfüllbarkeitstesters, noch eine Menge weiterer Optimierungsmöglichkeiten:

- Optimierungen der Kommunikation, z.Bsp. Caching, um eventuell mehrfache gleiche Anfragen an den FaCT-Server zu vermeiden;
- Optimierungen des Tableauablaufs, z.Bsp. mit Backjumping;
- Parallelisierung, z.Bsp. durch die Nutzung von Threads bei der Programmierung mit JAVA parallele Bearbeitung aller ODER-Zweige im Tableau möglich, dazu eventuell noch:
- die Nutzung mehrerer unabhängiger FaCT-Server.

Gerade die letzten zwei Punkte bieten ein weitreichendes Potential für weitere Entwicklungen. Denn der Ablauf des hier entwickelten Systems braucht nicht auf den Einsatz in Rechnernetzen mit nur einem FaCT-Server und einem SMDL-Client beschränkt werden. Aber wie schon angedeutet würde die Performance des Systems insbesondere durch die Programmierung mit Threads nicht nur in Rechnernetzen profitieren, sondern ebenso auf Multiprozessor-Rechnern.
Allerdings sind für diese Ausbaustufen umfangreiche Änderungen im System notwendig, welche sicherlich einer Neuprogrammierung gleichen würden.

Die ersten beiden Entwicklungsmöglichkeiten der am Anfang betrachteten Optimierungen sind für den Aufbau eines Wissensrepräsentationssystems (kurz WRS) basierend auf dem modularen Erfüllbarkeitstesters interessant. Denn für grössere Projekte bzw. Datenmengen ist jede Optimierung des Systems nötig. Gerade auf dem Gebiet der Tableaukalküle sind in dieser Richtung doch schon

verschiedene Möglichkeiten erörtert worden (siehe [4]). Diese sollten zum Aufbau eines WRS natürlich berücksichtigt werden.
Die notwendige Kommunikation der Teile des Systems untereinander bleibt auch weiterhin ein Ansatzpunkt für Verbesserungen. Denn solange zum Beispiel die Datenübertragung zwischen den Modulen den größten Rechenaufwand erfordert, haben Optimierung des Tableauablauf nur einen sehr geringen Einfluß auf die Gesamtperformance des Systems.
Allgemein steht deshalb bei Weiterentwicklungen die Suche nach den größten Verbrauchern von Rechenzeit des Systems im Vordergrund. Und danach sollte zuerst deren Optimierung vorangetrieben werden.

Diese Vorgehensweise bietet sich auch für die Implementation anderer modularer Systeme für ausdrucksstärkere Modellierungssprachen an. Denn der Ansatz, einen modularen Beweiser für die Kombination zweier Logiken zu schaffen, läßt sich, falls geeignete (Tableau-)Kalküle vorhanden sind, auch auf andere Logikkombinationen bzw. Modellierungssprachen übertragen.
Allein schon im Umfeld der Beschreibungs- und Modallogiken gibt es sehr viele einzelne Erweiterungen mit den entsprechenden Tableaus, die ihrerseits wieder durch weitere Kombination im Sinne eines modularen Beweisers profitieren würden.

Allerdings hat auch der Einsatz von modularen Beweisern seine Grenzen, denn nicht jede Erweiterung der Ausdrucksfähigkeiten ist noch durch geeignete Tableaus bzw. Algorithmen beherrschbar. Aber sicherlich sind bisher nur wenige Möglichkeiten erschlossen worden.

Anhang A

Dokumentation zur Implementation

A.1 Makros zum unimodalen Tableau

Folgende Makros sollten an entsprechender Stelle innerhalb der *solving*-Methode eingesetzt werden. Sie legen die Arbeitungsweise der *Solver*-Klasse bezüglich der einzelnen Formeltypen fest, und entsprechen damit einem Teil der Regeln des Tableaus.

Procedure 7 *proc_andNode*-Makro

```
1:  // Bearbeitung eines andNodes
2:
3:  childs = formula.getallChilds();
4:  for index = 0 to index < childs.size() do
5:     child = childs.getmyNode(index);
6:     if (child instanceof andNode) or (child instanceof orNode) then
7:        w.to_do_formulas.add(child);
8:     else if (child instanceof eq_topNode) or (child instanceof notNode)
       then
9:        w.clash_test_add(child);
10:    else if (child instanceof boxNode) then
11:       w.box_nodes_add(child);
12:       if reflexive_flag == true then
13:          w.to_do_formulas.add(child);
14:    else if (child instanceof diaNode) then
15:       w.dia_nodes_add(child);
16: if w.CLASH then
17:    reco_macro();
18:    continue; { Rücksprung zur äußeren while-Schleife und eventuell Ab-
       bruch.}
```

Procedure 8 *proc_ orNode*-Makro

```
1: // Bearbeitung eines orNodes
2:
3: w.SOLVE_INDEX = solving_index;
4: or_worlds.add_world(new World(w));
5: child_index=((orNode)formula).get_child_index();
6: childs = formula.getallChilds();
7: if child_index>=childs.size() then
8:    reco_macro();
9: else
10:   child = childs.getmyNode(child_index);
11:   if (child instanceof andNode) or (child instanceof orNode) then
12:      w.to_do_formulas.insert_add(child,solving_index+1);
13:   else if (child instanceof eq_topNode) or (child instanceof notNode)
      then
14:      w.clash_test_add(child);
15:   else if (child instanceof boxNode) then
16:      w.box_nodes_add(child);
17:      if reflexive_flag == true then
18:         w.to_do_formulas.add(child);
19:   else if (child instanceof diaNode) then
20:      w.dia_nodes_add(child);
21:   if w.CLASH then
22:      reco_macro();
23:      continue; { Rücksprung zur äußeren while-Schleife und eventuell Ab-
         bruch.}
```

Procedure 9 *proc_ notNode-* und *proc_ eqtopNode*-Makro

```
1: // Bearbeitung eines notNodes oder eines eq_toptNodes.
2:
3: clash_test_add(w,formula);
4: if w.CLASH then
5:    reco_macro();
6:    continue; { Rücksprung zur äußeren while-Schleife und eventuell Ab-
      bruch.}
```

Procedure 10 *proc_ diaNode*-Makro

```
1: // Bearbeitung eines diaNodes. Eigentliche Bearbeitung erfolgt in einer ex-
   tra Schleife direkt innerhalb der solving-Methode
2:
3: w.dia_nodes.add(formula);
```

Procedure 11 *proc_ boxNode*-Makro

```
1: // Bearbeitung eines boxNodes. Eigentliche Bearbeitung erfolgt in einer extra Schleife direkt innerhalb der solving-Methode
2:
3: w.box_nodes.add(formula);
4: childs = formula.getallChilds();
5: child = childs.getmyNode(0);
6: if reflexive_flag == true then
7:    w.to_do_formulas.add(child);
```

Procedure 12 *reco_ macro*-Makro

```
1: {Makro zur Rekonstruktion der Weltenstruktur nach Auftreten eines Clash oder negativen ALC-Test.}
2: w = reco_world(w);
3: if w!=null then
4:    solving_index = w.SOLVE_INDEX;
5:    formula = w.to_do_formulas.getmyNode(solving_index);
6:    child_index = ((orNode)formula).get_child_index();
7:    child_index = child_index + 1;
8:    ((orNode)formula).set_child_index(child_index);
9: else
10:   sat = false;
```

A.2 Quellen des polymodalen Tableau

Nachfolgend werden die neuen bzw. erweiterten Methoden *check_ready()*, *add_to_reachable_worlds()* und *blocking_world()* vorgestellt.
Die erste Methode wird zur Steuerung des Ablaufs des polymodalen Tableaus benutzt, und stellt eine zusätzliche Methode im Aufbau der modifizierten *Solver*-Klasse dar.
Innerhalb der Quelltextabschnitte *Procedure 14* und *Procedure 15* werden die veränderten Methoden der *Solver*-Klasse beschrieben.
Die Methode *add_to_reachable_worlds* unterscheidet sich insbesondere von der einfacheren Version [1] des unimodalen Tableaus durch die zusätzliche `if`-Bedingung (ab Codezeile 12) nach der `for`-Schleife.
In beiden Erweiterungen werden die Hauptveränderungen des Quelltextes durch zusätzliche Vergleiche (Codezeile 9 und 12 bzw. 20) der `PARENT_INDEX`-Werte der verschiedenen Welten hervorgerufen.
Diese Vergleiche realisieren die zusätzlichen Bedingungen beim Blockingtest und der Ausführung der Regel für die Box, das heißt es werden der Bezeichner der Kante zwischen zwei Welten[2] und der

[1] bisher nicht angegeben
[2] entspricht `w.PARENT_INDEX`

Bezeichner der entsprechenden Box[3] verglichen.

Procedure 13 zusätzliche *check_ready*-Methode

```
 1: {Aufruf: boolean check_ready(World w)}
 2: // Diese Methode überprüft rekursiv die gesamte Weltenstrukur, ob jede
 3: // Welt vollständig bearbeitet wurde (, d.h. w.SOLVED==true).
 4: // liefert: false sobald eine Welt noch weiterbearbeitet werden muss.
 5: boolean res = false;
 6:
 7: res = w.SOLVED;
 8: if res then
 9:    for index = 0 to (w.reachable_worlds.size()-1) do
10:       res = check_ready(w.reachable_worlds.get_world(index));
11:       if res==false then
12:          break;
13: return res;
```

Procedure 14 modifizierte *add_to_reachable_worlds*-Methode

```
 1: {Aufruf: void add_to_reachable_worlds(World box_w, boxNode bnode)}
 2: // Diese Methode erweitert die Formelmengen neu eingeführter Welten
 3: // und der Vorgängerwelt bei Abarbeitung der boxNodes.
 4: World w_for_add = null;
 5: int p_index = bnode.get_box_id();
 6:
 7: for index = 0 to (box_w.reachable_worlds.size()-1) do
 8:    w_for_add = box_w.reachable_worlds.get_world(index);
 9:    if w_for_add.PARENT_INDEX == p_index then
10:       w_for_add.to_do_formulas.add((myNode)bnode);
11:       w_for_add.SOLVED = false;
12: if box_w.parent_world.PARENT_INDEX == p_index then
13:    box_w.parent_world.to_do_formulas.add((myNode)bnode);
14:    box_w.parent_world.SOLVED = false;
```

[3] entspricht `bnode.get_box_id()`

Procedure 15 modifizierte *blocking_ world*-Methode

```
1: {Aufruf: boolean blocking_world(World blocked_w, World blocking_w)}
2: // Diese Methode realisiert den Blocking-Test des polymodalen Tableaus.
3: // liefert: true falls Blocking-Test positiv (d.h. blocking_w blockiert
   blocked_w), false sonst
4: boolean block = false;
5: boolean exists = true;
6: myNode child = null;
7:
8: if blocking_w == null then
9:    block = false;
10: else
11:    for index = 0 to (blocked_w.box_nodes.size()-1) do
12:       child = blocked_w.box_nodes.getmyNode(index);
13:       exists = exists and blocking_w.box_nodes.exist_Node(child.getID());
14:    for index = 0 to (blocked_w.dia_nodes.size()-1) do
15:       child = blocked_w.dia_nodes.getmyNode(index);
16:       exists = exists and blocking_w.dia_nodes.exist_Node(child.getID());
17:    if exists then
18:       block = true;
19:    else
20:       if blocked_w.PARENT_INDEX == blocking_w.PARENT_INDEX
          then
21:          block = blocking_world(blocked_w, blocking_w.parent_world);
22:       else
23:          block = false;
24: return block;
```

Anhang B

XML-Definition der Syntax

Syntaxdefinition in XML mittels der Datei polyS5.dtd,
modallogischer Teil:

```
<!– dtd for polyS5 formulas –>

<!ELEMENT   FORMULA (CONCEPT|NOT|BOX|DIA|AND|OR|IMP)>

<!ELEMENT   NOT   (CONCEPT|NOT|BOX|DIA|AND|OR|IMP)>

<!ELEMENT   BOX   (BOX_NAME, BOX_FORMULA)>

<!ELEMENT   BOX_FORMULA
            (CONCEPT|NOT|BOX|DIA|AND|OR|IMP)>

<!ELEMENT   BOX_NAME   EMPTY>
<!ATTLIST    BOX_NAME   NAME   CDATA   #REQUIRED>

<!ELEMENT   DIA   (DIA_NAME, DIA_FORMULA)>

<!ELEMENT   DIA_FORMULA
            (CONCEPT|NOT|BOX|DIA|AND|OR|IMP)>

<!ELEMENT   DIA_NAME   EMPTY>
<!ATTLIST    DIA_NAME   NAME   CDATA   #REQUIRED>

<!ELEMENT   AND   (CONCEPT|NOT|BOX|DIA|AND|OR|IMP)+>

<!ELEMENT   OR   (CONCEPT|NOT|BOX|DIA|AND|OR|IMP)+>

<!ELEMENT   IMP   ((CONCEPT|NOT|BOX|DIA|AND|OR|IMP),
                   (CONCEPT|NOT|BOX|DIA|AND|OR|IMP))>
```

Syntaxdefinition in XML mittels der Datei polyS5.dtd,
beschreibungslogischer Teil:

```
<!ELEMENT   CONCEPT
    (PRIMITIVE|TOP|BOTTOM|NOT_A|AND_A|OR_A|SOME|ALL)>

<!ELEMENT   NOT_A
    (PRIMITIVE|TOP|BOTTOM|NOT_A|AND_A|OR_A|SOME|ALL)>

<!ELEMENT   AND_A
    (PRIMITIVE|TOP|BOTTOM|NOT_A|AND_A|OR_A|SOME|ALL)+>

<!ELEMENT   OR_A
    (PRIMITIVE|TOP|BOTTOM|NOT_A|AND_A|OR_A|SOME|ALL)+>

<!ELEMENT   PRIMITIVE   EMPTY>
<!ATTLIST    PRIMITIVE   NAME   CDATA   #REQUIRED>

<!ELEMENT   SOME   ((PRIMROLE),
    (PRIMITIVE|TOP|BOTTOM|NOT_A|AND_A|OR_A|SOME|ALL))>

<!ELEMENT   ALL   ((PRIMROLE),
    (PRIMITIVE|TOP|BOTTOM|NOT_A|AND_A|OR_A|SOME|ALL))>

<!ELEMENT   PRIMROLE   EMPTY>
<!ATTLIST    PRIMROLE   NAME   CDATA   #REQUIRED>

<!ELEMENT   TOP   EMPTY>

<!ELEMENT   BOTTOM   EMPTY>
```

Anhang C

Inhalt der beigelegten CD

- Verzeichnis : `diplom` - diese Diplomarbeit im Postskript- (`dipl.ps`) und im dvi-Format (`dipl.dvi`)
- Verzeichnis : `jsolver` - Source Code Verzeichnis, `*.java` und `*.class` Dateien und entsprechende package-Unterverzeichnisse
  - Unterverzeichnis : `jsolver/main/xml` - `xml`-Formel Beispiele, alle Benchmarkformeln für K, T und $S4$ sowie einige einfache Beispiele für $S5_n$
- Verzeichnis : `doc` - Dokumentation des Source Code
  - Unterverzeichnis : `doc/html` - Dokumentation im html-Format,(Datei `index.html` mit einem html-fähigem Browser laden)
  - Unterverzeichnis : `doc/tex` - Dokumentation im LaTeX-Format (Datei `mydoc.tex`)
- Verzeichnis : `xml-parser` - Datei `Parser.txt` beschreibt Bezug und Installation des XML-Parsers für Java
- Verzeichnis : `fact` - Datei `FaCT.txt` beschreibt Bezug und Installation des FaCT-Servers

Abbildungsverzeichnis

Quellcodeverzeichnis

Literaturverzeichnis

[1] Ian R. Horrocks: *Optimising tableaux decision procedures for description Logics* , 1997 , Dissertation Uni Manchester

[2] Rajeev Goré: *Tableau Methods for Modal and Temporal Logics*, in *Handbook of Tableau Methods* editors M.D. Agostino, D.Gabbay, R. Hähnle and J. Posegga, 1998 Kluwer, Dordrecht

[3] K.Schild: *A correspondence theory for terminological logics:* Preliminary report, 1991, in *Proceedings of IJCAI-91*, 40:466-471

[4] I.R. Horrocks, P.F. Patel-Schneider: *Optimising Description Logic Subsumption*, June 1999, in *Journal of Logic and Computation*, Vol.9 No.3:267-293, Special Issue: Description Logics

[5] Peter Balsiger, Alain Heuerding, Stefan Schwendimann: *A benchmark method for the propositional modal logics K, KT, S4* , 1999 , Universität Bern

[6] Franz Baader, Armin Laux: *Terminological Logics with Modal Operators* , in *C.S.Mellish(ed.), Proc. of the 14th International Joint Conference on Artifical Intelligence* , 1995, Morgan Kaufmann

[7] Ronald Fagin, Joseph Y. Halpern, Yoram Moses, Moshe Y. Vardi: *Reasoning about Knowledge* , MIT Press

[8] K.J.J. Hintikka: *Knowledge and Belief. An Introduction to the Logic of the Two Notions* , Cornell Univ. Press, 1962

[9] D. Pearce: *Epistemic Operators and Knowledge-Based Reasoning* , in *Knowledge and Belief in Philosophy and Artificial Intelligence* LOGICA NOVA Akademie Verlag

[10] M. Schmidt-Schaußs, G. Smolka: *Attributive concept descriptions with complements* , in *Artifical Intelligence* , 48:1-26 , 1991

[11] A. Borgida, *Description logics in data management* , IEEE Transaction on Knowledge and Data Engineering, 7(5) Oct. 1995

[12] A. Borgida, *On the relative expressiveness of description logics and first order logics* , in *Artifical Intelligence* , 82:353-367 , 1996

[13] C. Bettini, *Time-dependent concepts: representation and reasoning using temporal description logics* , DATA & KNOWLEDGE ENGINEERING 22 , 1997

[14] A.Borgida, R.J.Brachman, D.L. McGuiness and L. Alperin Resnick, *CLASSIC: A structural data model for objects* , ACM SIGMOND RECORD, 18(2) Jun. 1989

[15] R.J. Brachman, D.L. McGuiness, P.F. Patel-Schneider, L. Alperin Resnick and A.Borgida, *LIVING WITH CLASSIC: When and How to Use a KL-ONE-Like Language*

[16] R.J. Brachman, R.E. Fikes and H.J. Levesque, *KRYPTON: A functional approach to knowledge representation* , IEEE Computer, 16(10), Oct. 1983

[17] F. Baader, B. Hollunder. *KRIS: Knowledge representation and inference system* , SIGART Bulletin, 2(3):8-14 , 1991

[18] R.J. Brachman and G. Schmolze, *An Overview of the KL-ONE knowledge representation system* , Cognitive Science, 9(2), Apr. 1985

[19] A. Kobsa, *First experiences with the SB-ONE knowledge representation workbench in natural-language applications*, SIGART Bulletin, 2(3):70-76 , 1991

[20] D.B. Lenat and R.V. Guha *The evolution of CycL, the Cyc representation language* , SIGART Bulletin, 2(3):84-87 , 1991

[21] R.M. MacGregor *Inside the LOOM description classifier* , SIGART Bulletin, 2(3):88-92 , 1991

[22] E.Mays, R. Dionne and R. Weida *K-rep system overview* , SIGART Bulletin, 2(3):93-97 , 1991

[23] M.G. Moser *An overview of NIKL: the new implementation of KL-ONE* , Technical report 5421, Bolt, Beranek and Newman, Cambridge, MA, 1983

[24] S. Navathe, A. Savasere, T. Anwar, H. Beck and S. Gala , *Object modelling using classification in CANDIDE and its applications* , in A. Dogac, T. Ozsu, A. Briliris and T. Sellis, editors, *Advances in Object-Oriented Database Systems* , NATO ASI Series Vol 130, 1994

[25] C. Peltason *The BACK system - an overview* , SIGART Bulletin, 2(3):114-119 , 1991

[26] P.F. Patel-Schneider, *The CLASSIC knowledge representation system: Guiding principal and implementation rationale* , SIGART Bulletin, 2(3):108-113 , 1991

[27] A. Rector, S. Bechhofer, C.A. Goble, I. Horrocks, W.A. Nowlan and W.D. Solomon , *The GRAIL concept modelling language for medical terminology* , Artifical Intelligence in Medicine , 9:139-171, 1997

[28] W.A. Woods and J.G. Schmolze , *The KL-ONE family* , Computers and Mathematics with Applications - Spezial Issue on Artifical Intelligence , 23(2-5): 133-177, 1992

[29] B. Nebel , *Reasoning and Revision in Hybrid Representation Systems* , Number 422 in Lecture Notes in Artifical Intelligence, Springer-Verlag, Berlin, 1990

[30] P.F. Patel-Schneider , *System description: DLP* , Bell Labs Research, Murray Hill, NJ, Dec. 1997

[31] F. Giunchiglia, R. Sebastiani: *A SAT-based decision procedure for ALC*, in *Principles of Knowledge Repressentation and Reasoning*, Proc. of 5th International Conference (KR'96), L.C. Aiello, J. Doyle and S.C. Shapiro, eds., 304-314, Morgan Kaufmann, San Francisco, CA, 1996

[32] L. Kreiser, S. Gottwald, W. Stelzner , (Herausgeber) , *Nichtklassiche Logik, Eine Einführung* , Akademie Verlag Berlin , 1990

[33] SAX (Simple API to XML), Java Project X Core Library, The Java Developer Connection, `http://java.sun.com/jdc`

[34] S. Fischer, W. Müller, *Netzwerkprogrammierung unter LINUX und UNIX*, Carl Hanser Verlag München Wien, 1996

[35] A Client/Server - FaCT - System, CORBA-FaCT `http://www.cs.man.ac.uk/~horrocks/FaCT`

[36] JDK for Linux, http://www.blackdown.org/

[37] Allegro-Lisp, http://www.franz.com/

Erklärung

Ich versichere, daß ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig, den 15. März 2001 .....

Thomas Hofmann